Intel® Pentium® II Xeon™ Processor Specification Update

Release Date: December 2000

Order Number: 243776-027

The Pentium® II Xeon processor may contain design defects or errors known as errata that may cause the product to deviate from published specifications. Current characterized errata are documented in this Specification Update.

CONTENTS

REVISION HISTORY

Date of Revision	Version	Description
May 1998	-001	This document is the first Specification Update for the Pentium® II Xeon™ processor.
June 1998	-002	Updated Erratum 29. Added Erratum 34, Specification Clarifications 8 through 13, and Documentation Changes 13 through 16.
July 1998	-003	Launch edition of the Pentium II Xeon processor Specification Update. Added Errata 35 through 37.
August 1998	-004	Added Processor Markings. Updated Errata 6, 29, and 34 through 37. Added Erratum 38. Updated Specification Clarification 4.
September 1998	-005	Updated Pentium II Xeon Processor Identification and Package Information table. Added B1 stepping information. Added Specification Change 1. Updated Errata 35 and 36. Added Errata 39 through 42.
October 1998	-006	Updated Pentium II Xeon Processor Identification and Package Information table. Modified entry numbering. Updated Errata 1 and 27. Added Specification Changes 2 and 3. Added Errata 43 through 45. Added Specification Clarifications 14 and 15.
November 1998	-007	Updated Pentium II Xeon Processor Identification and Package Information table. Updated Errata status in Summary Table of Changes. Updated Erratum D31. Added Specification Change D4. Added Errata D46 and D47. Added Specification Clarification D16. Updated Documentation Change D10. Added Documentation Change D17.
December 1998	-008	Updated Specification Change D2. Added Erratum D48. Added Specification Change D5. Updated processor identification table.
January 1999	-009	Added Specification Change D6. Added Errata D49 through D52. Added Documentation Changes D18 through D20. Updated processor identification table.
February 1999	-010	Updated processor identification table. Added Erratum D53.
March 1999	-011	Added Specification Change D3. Added S-Spec Definition. Removed Specification Changes, Specification Clarifications, and Document Changes that have been incorporated into the appropriate documentation. Renumbered remaining items.
April 1999	-012	Moved revised Mixed Steppings statement to the General Information section and renumbered remaining items.
June 1999	-013	Added Erratum D54. Added Documentation Change D1. Added Specification Clarifications D2 and D3. Added Specification Change D3.
July 1999	-014	Added Erratum D55. Updated status for Errata D25, D34, D35, D36 and D49.

REVISION HISTORY

Date of Revision	Version	Description
August 1999	-015	Added Document Change D2. Updated Codes Used in Summary Table. Updated column heading in Errata, Documentation Changes, Specification Clarifications and Specification Changes tables.
October 1999	-016	Added Brand ID column to *Identification Information*. Added Erratum D56.
November 1999	-017	Added Erratum D57 and D58. Added Documentation Change D3.
December 1999	-018	Added Erratum D59. Added Documentation Change D4.
January 2000	-019	Revised Erratum D59. Added Errata D60 and D61. Added Documentation Change D5.
February 2000	-020	Revised Erratum D58. Added Documentation Change D6. Updated Summary of Changes product letter codes.
March 2000	-021	Added Erratum D62.
July 2000	-022	Revised Erratum D52. Added Errata D63 and D64.
August 2000	-023	Added Erratum D65. Added Specification clarification D4.
September 2000	-024	Revised Errata D38, D57, and D61. Added Errata D66 and D67. Added Documentation Changes D7 and D8.
October 2000	-025	Updated documents in Preface. Added Erratum D68. Added Documentation Changes D9 and D10.
November 2000	-026	Added Erratum D69.
December 2000	-027	Updated Specification Update product key to include the Intel® Pentium® 4 processor, Revised Erratum D2. Added Documentation Changes D11 through D16.

PREFACE

This document is an update to the specifications contained in the following documents:

- *Pentium® II Xeon™ Processor at 400 and 450 MHz* datasheet (Order Number 243770)
- *Intel Architecture Software Developer's Manual*, *Volumes 1, 2 and 3* (Order Numbers 243190, 243191, and 243192, respectively)
- *P6 Family of Processors Hardware Developer's Manual* (Order Number 244001)

It is intended for hardware system manufacturers and software developers of applications, operating systems, or tools. It contains S-Specs, Errata, Documentation Changes, Specification Clarifications and, Specification Changes.

Nomenclature

S-Spec Number is a five-digit code used to identify products. Products are differentiated by their unique characteristics, e.g., core speed, L2 cache size, package type, etc. as described in the processor identification information table. Care should be taken to read all notes associated with each S-Spec number.

Errata are design defects or errors. Errata may cause the Pentium II Xeon processor's behavior to deviate from published specifications. Hardware and software designed to be used with any given processor must assume that all errata documented for that processor are present on all devices unless otherwise noted.

Documentation Changes include typos, errors, or omissions from the current published specifications. These changes will be incorporated in the next release of the specifications.

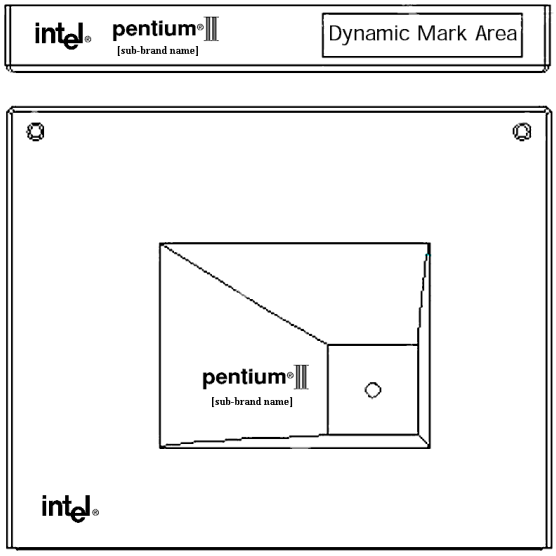
Specification Clarifications describe a specification in greater detail or further highlight a specification's impact to a complex design situation. These clarifications will be incorporated in the next release of the specifications.

Specification Changes are modifications to the current published specifications for the Pentium® II Xeon™ processor. These changes will be incorporated in the next release of the specifications.

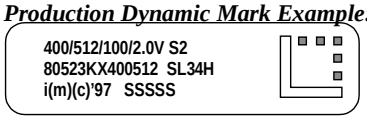
Specification Update for the Pentium® II Xeon™ Processor

GENERAL INFORMATION

Pentium® II Xeon™ and Boxed Pentium® II Xeon™ Processor Markings

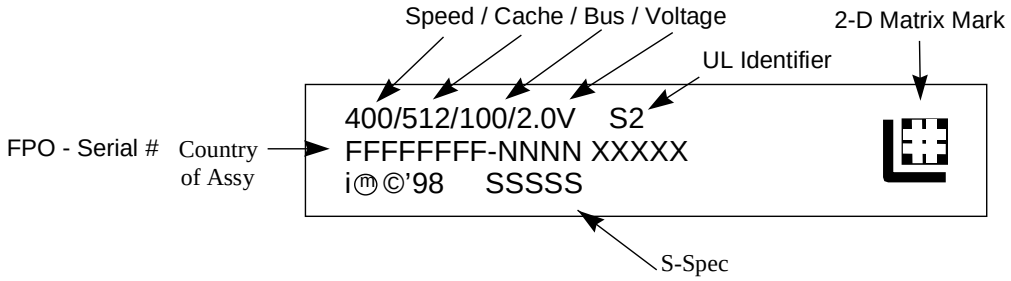


Production Dynamic Mark Example:



2D Matrix Contents Example:
Intel 80523KX400512
FFFFFFFF-NNNN

Dynamic Mark Area



IDENTIFICATION INFORMATION

The Pentium II Xeon processor can be identified by the following values:

Family[1]	400 and 450 MHz Pentium® II Xeon™ Processor[2]	Brand ID[3]
0110	0101	00h = Not Supported

NOTES:

1. The Family corresponds to bits [11:8] of the EDX register after Reset, bits [11:8] of the EAX register after the CPUID instruction is executed with a 1 in the EAX register, and the generation field of the Device ID register accessible through Boundary Scan.
2. The Model corresponds to bits [7:4] of the EDX register after Reset, bits [7:4] of the EAX register after the CPUID instruction is executed with a 1 in the EAX register, and the model field of the Device ID register accessible through Boundary Scan.
3. The Brand ID corresponds to bits [7:0] of the EBX register after the CPUID instruction is executed with a 1 in the EAX register.

The Pentium II Xeon processor's second level (L2) cache size can be determined by the following register contents

512-Kbyte Unified L2 Cache[1]	43h
1-Mbyte Unified L2 Cache[1]	44h
2-Mbyte Unified L2 Cache[1]	45h

NOTES:

1. For the Pentium® II Xeon™ processor, the unified L2 cache size corresponds to a token in the EDX register after the CPUID instruction is executed with a 2 in the EAX register. Other Intel microprocessor models or families may move this information to other bit positions or otherwise reformat the result returned by this instruction; generic code should parse the resulting token stream according to the definition of the CPUID instruction.

Mixed Steppings in MP Systems

Intel Corporation fully supports mixed steppings of Pentium II Xeon processors. The following list and processor matrix describes the requirements to support mixed steppings:

- While Intel has done nothing to specifically prevent processors operating at differing frequencies from functioning within a multiprocessor system, there may be uncharacterized errata that exist in such configurations. Intel does not support such configurations. In mixed stepping systems, all processors must operate at identical frequencies (i.e., the highest frequency rating commonly supported by all processors).
- While there are no known issues associated with the mixing of processors with differing cache sizes in a multiprocessor system, and Intel has done nothing to specifically prevent such system configurations from operating, Intel does not support such configurations since there may be uncharacterized errata that exist. In mixed stepping systems, all processors must be of the same cache size.
- While Intel believes that certain customers may wish to perform validation of system configurations with mixed frequency or cache sizes, and that those efforts are an acceptable option to our customers, customers would be fully responsible for the validation of such configurations.
- The workarounds identified in this and following specification updates must be properly applied to each processor in the system. Certain errata are specific to the multiprocessor environment and are identified in the *Mixed Stepping Processor Matrix* found at the end of this section. Errata for all processor steppings will affect system performance if not properly worked around. Also see the "Pentium® II Xeon™ Processor Identification and Package Information" section for additional details on which processors are affected by specific errata.
- In mixed stepping systems, the processor with the lowest feature-set, as determined by the CPUID Feature Bytes, must be the Bootstrap Processor (BSP). In the event of a tie in feature-set, the tie should be resolved by selecting the BSP as the processor with the lowest stepping as determined by the CPUID instruction.
- *Functional Redundancy Checking Mode* (FRC mode) is not supported using a master and checker pair of processors with different stepping, model number, cache size, or frequency.

In the following processor matrix, "NI" indicates that there are currently no known issues associated with mixing these steppings. A number indicates that a known issue has been identified as listed in the table following the matrix. A multiprocessor system using mixed processor steppings must assure that errata are addressed appropriately for each processor.

Mixed Stepping Processor Matrix

Stepping	B0	B1
B0	1,2	1,2
B1	1,2	1

NOTES:

1. Some of these processors are affected by errata which may affect the features an MP system is able to support. See the "Pentium® II Xeon™ Processor Identification and Package Information" table for details on which processors are affected by these errata.
2. Some B0 stepping processors must be operated at a lower T_{PLATE} specification than normal (65 °C). See the "Pentium® II Xeon™ Processor Identification and Package Information" table for more details.

Pentium® II Processor Identification Information

S-Spec	Core Steppings	CPUID	Speed (MHz)	L2 Size (Kbytes)	GC82459AA (C6C) Stepping	Processor Substrate Revision	Cartridge Revision	Notes
SL2NB	B0	0652h	400	1024	A3	1M/2M-Pf	2.0	1, 2, 3, 4, 6
SL2RH	B0	0652h	400	512	A3	512K-Pb	2.0	1, 2, 3, 4, 6
SL2XJ	B1	0653h	450	512	B0	512K-Oa	2.0	7, 9, 10
SL2XK	B1	0653h	450	1024	B0	1M/2M-Oa	2.0	7, 9, 10
SL2XL	B1	0653h	450	2048	B0	1M/2M-Oa	2.0	7, 9, 10
SL33T	B1	0653h	450	512	B0	512K-oA	2.0	4, 7, 8, 9, 10
SL344	B0	0652h	400	512	A3	512K-pB	2.0	1, 2, 3, 5, 6
SL345	B0	0652h	400	1024	A3	1M/2M-pF	2.0	1, 2, 3, 5, 6
SL34H	B1	0653h	400	512	A3	512K-pB	2.0	1, 2, 5, 6
SL34J	B1	0653h	400	1024	A3	1M/2M-pF	2.0	1, 2, 5, 6
SL354	B1	0653h	450	512	B0	512K-oA	2.0	4, 7, 9, 10
SL35N	B1	0653h	400	512	A3	512K-pB	2.0	1, 2, 5, 6, 8
SL35P	B1	0653h	400	1024	A3	1M/2M-pF	2.0	1, 2, 5, 6, 8
SL36W	B1	0653h	450	512	B0	512K-oA	2.0	7, 8, 9, 10
SL33U	B1	0653h	450	1024	B0	1M/2M-oA	2.0	7, 8, 9, 10
SL33V	B1	0653h	450	2048	B0	1M/2M-oA	2.0	7, 8, 9, 10

NOTES:

1. These processors will not shut down automatically on assertion of THERMTRIP#.
2. These processors are affected by Erratum D30.
3. These Processor Information ROM in these processors contain the S-spec number, but the leading “S” character is replaced with a space character.
4. These processors are affected by Erratum D36.
5. These processors must be operated at a T_{PLATE} specification of 65 ºC.
6. These processors require an SMBus input setup time (T57) of 800 ns.
7. Error Checking and Correcting (ECC) for the L2 cache transactions cannot be disabled on these processors.
8. This is a boxed processor with attached passive heatsink See Specification Change D6 of this document for the boxed processor specification.
9. These processors are affected by Erratum D49.
10. These processors are affected by Erratum D50.

SUMMARY OF CHANGES

The following table indicates the Errata, Documentation Changes, Specification Clarifications, or Specification Changes that apply to Pentium II processors. Intel intends to fix some of the errata in a future stepping of the component, and to account for the other outstanding issues through documentation or specification changes as noted. This table uses the following notations:

CODES USED IN SUMMARY TABLE

X:	Erratum, Documentation Change, Specification Clarification or Specification Change applies to the given processor stepping.
(No mark) or (blank box):	This item is fixed in or does not apply to the given stepping.
Fix:	This erratum is intended to be fixed in a future stepping of the component.
Fixed:	This erratum has been previously fixed.
NoFix:	There are no plans to fix this erratum.
Doc:	Intel intends to update the appropriate documentation in a future revision.
PKG:	This column refers to errata on the Pentium II processor substrate.
AP:	APIC related erratum.
Shaded:	This item is either new or modified from the previous version of the document.

Each Specification Update item is prefixed with a capital letter to distinguish the product. The key below details the letters that are used in Intel's microprocessor Specification Updates:

A = Intel® Pentium® II processor

B = Intel® Mobile Pentium® II processor

C = Intel® Celeron™ processor

D = Intel® Pentium® II Xeon™ processor

E = Intel® Pentium® III processor

G = Intel® Pentium® III Xeon™ processor

H = Intel® Mobile Celeron™ processor at 466 MHz, 433 MHz, 400 MHz, 366 MHz, 333 MHz, 300 MHz, and 266 MHz

K = Intel® Mobile Pentium® III processor

M = Intel® Mobile Celeron™ processor at 500 MHz, 450 MHz, and 400A MHz

N = Intel® Pentium® 4 processor

The Specification Updates for the Pentium® processor, Pentium® Pro processor, and other Intel products do not use this convention.

Summary of Errata

NO.	B0	B1	PKG	Plans	ERRATA
D1	X	X		NoFix	FP data operand pointer may be incorrectly calculated after FP access which wraps 64-Kbyte boundary in 16-bit code
D2	X	X		NoFix	Differences exist in debug exception reporting
D3	X	X		NoFix	FLUSH# servicing delayed while waiting for STARTUP_IPI in MP systems
D4	X	X		NoFix	Code fetch matching disabled debug register may cause debug exception
D5	X	X		NoFix	Double ECC error on read may result in BINIT#
D6	X	X		NoFix	FP inexact-result exception flag may not be set
D7	X	X		NoFix	BTM for SMI will contain incorrect FROM EIP
D8	X	X		NoFix	I/O restart in SMM may fail after simultaneous MCE
D9	X	X		NoFix	Branch traps do not function if BTMs are also enabled
D10	X	X		NoFix	Checker BIST failure in FRC mode not signaled
D11	X	X		NoFix	BINIT# assertion causes FRCERR assertion in FRC mode
D12	X	X		NoFix	Machine check exception handler may not always execute successfully
D13	X	X		NoFix	LBER may be corrupted after some events
D14	X	X		NoFix	BTMs may be corrupted during simultaneous L1 cache line replacement
D15	X	X		NoFix	A20M# my be inverted after returning from SMM and Reset
D16	X	X		NoFix	Near CALL to ESP creates unexpected EIP address
D17	X	X		NoFix	Mixed cacheability of lock variables problematic in MP systems
D18	X	X		NoFix	MCE due to L2 parity error gives L1 MCACOD.LL
D19	X	X		NoFix	Memory type field undefined for nonmemory operations
D20	X	X		NoFix	Infinite snoop stall during L2 initialization of MP systems
D21	X	X		NoFix	FP data operand pointer may not be zero after power on or reset
D22	X	X		NoFix	Premature execution of a load operation prior to exception handler invocation
D23	X	X		NoFix	EFLAGS discrepancy on page fault after multiprocessor TLB shootdown
D24	X	X		NoFix	Read portion of RMW instruction may execute twice
D25	X	X		NoFix	Test pin must be high during power up

Summary of Errata

NO.	B0	B1	PKG	Plans	ERRATA
D26			X	Fixed	Processor Information ROM uses WP pin
D27	X	X		NoFix	Intervening writeback may occur during locked transaction
D28	X	X		NoFix	MC2_STATUS MSR has model-specific error code and Machine Check Architecture error code reversed
D29	X	X		NoFix	Cache access while changing BBL_CR_CTL3 configuration may cause hang
D30			X	Fix	Thermal sensor may assert SMBALERT# incorrectly
D31	X	X		NoFix	MOVD following zeroing instruction can cause incorrect result
D32	X	X		NoFix	Top 4 PAT entries not usable with Mode B or Mode C paging
D33	X	X		NoFix	MOV with debug register causes debug exception
D34	X	X		NoFix	UC write may be reordered around a cacheable write
D35	X	X		NoFix	Misprediction in program flow may cause unexpected instruction execution
D36	X	X		Fixed	System bus ECC may report false errors
D37	X			Fixed	Full in order queue may cause infinite DBSY# assertion
D38	X	X		NoFix	Data breakpoint exception in a displacement relative near call may corrupt EIP
D39	X	X		NoFix	Fault on REP CMPS/SCAS operation may cause incorrect EIP
D40	X	X		NoFix	System bus ECC not functional with 2:1 ratio
D41	X	X		NoFix	RDMSR or WRMSR to invalid MSR address may not cause GP fault
D42	X			Fixed	Null selectors may cause FRC errors in FRC-enabled systems
D43	X	X		NoFix	SYSENTER/SYSEXIT instructions can implicitly load “null segment selector” to SS and CS registers
D44	X	X		NoFix	PRELOAD followed by EXTEST does not load boundary scan data
D45	X	X		NoFix	Far jump to new TSS with D-bit cleared may cause system hang
D46	X	X		NoFix	Illegal opcode during L2 cache initialization
D47	X	X		NoFix	Incorrect chunk ordering may prevent execution of the Machine Check Exception handler after BINIT#
D48	X	X		NoFix	Resume flag may not be cleared after debug exception
D49				Fixed	Thermal sensor leakage current may exceed specification
D50	X	X		NoFix	System bus address parity checking may report false AERR#
D51	X	X		NoFix	Misaligned locked access to APIC space results in hang

Summary of Errata

NO.	B0	B1	PKG	Plans	ERRATA
D52	X	X		NoFix	Potential loss of data coherency during MP data ownership transfer
D53	X	X		NoFix	Memory ordering based synchronization may cause a livelock condition in MP systems
D54	X	X		NoFix	GP# fault on WRMSR to ROB_CR_BKUPTMPDR6
D55	X	X		NoFix	Machine check exception may occur due to improper line eviction in the IFU
D56	X	X		NoFix	Lower bits of SMRAM SMBASE register cannot be written with an ITP
D57	X	X		NoFix	Task switch caused by page fault may cause wrong PTE and PDE access bit to be set
D58	X	X		NoFix	Unsynchronized cross-modifying code operations can cause unexpected instruction execution results
D59	X	X		NoFix	Deadlock may occur due to illegal-instruction/page-miss combination
D60	X	X		NoFix	FLUSH# assertion following STPCLK# may prevent CPU clocks from stopping
D61	X	X		NoFix	Floating-point exception condition may be deferred
D62			X	NoFix	Race conditions may exist on Thermal Sensor SMBus Collision Detection/Arbitration Circuitry
D63	X	X		NoFix	Selector for the LTR/LLDT register may get corrupted.
D64	X	X		NoFix	Init does not clear global entries in the TLB.
D65	X	X		NoFix	VM bit will be cleared on a double fault handler.
D66	X	X		NoFix	Memory aliasing with inconsistent A and D bits may cause processor deadlock
D67	X	X		NoFix	Use of memory aliasing with inconsistent memory type may cause system hang
D68	X	X		NoFix	Processor may report invalid TSS fault instead of double fault during mode C paging
D69	X	X		NoFix	Machine check exception may occur when interleaving code between different memory types
D1AP	X	X		NoFix	APIC access to cacheable memory causes shutdown
D2AP	X	X		NoFix	MP systems may hang due to catastrophic errors during BSP determination
D3AP	X	X		NoFix	Write to mask LVT (programmed as EXTINT) will not deassert outstanding interrupt

Summary of Documentation Changes

NO.	B0	B1	PKG	Plans	DOCUMENTATION CHANGES
D1	X	X		Doc	STPCLK# pin definition
D2	X	X		Doc	Invalidating caches and TLBs
D3	X	X		Doc	Handling of self-modifying and cross-modifying code
D4	X	X		Doc	Machine check architecture initialization of MCi_STATUS registers
D5	X	X		Doc	LOCK# signal prefix operands
D6	X	X		Doc	SMRAM state save map contains documentation errors
D7	X	X		Doc	System Management Interrupt (SMI) during startup IPI clarification
D8	X	X		Doc	Memory aliasing with different memory types
D9	X	X		Doc	RUNBIST will not function when STPCLK# driven low
D10	X	X		Doc	Memory aliasing with inconsistent A and D bits may cause processor deadlock
D11	X	X		Doc	An interrupt could occur while TSS is marked busy
D12	X	X		Doc	NMI unmasked early when processor is running in V86 mode
D13	X	X		Doc	P6 reads two bytes for POP SEG instruction
D14	X	X		Doc	APIC register offsets are aligned on 128-bit boundaries
D15	X	X		Doc	Single stepping of instructions breaks out of HALT state
D16	X	X		Doc	Additional signal resumes execution while in a HALT state

Summary of Specification Clarifications

NO.	B0	B1	PKG	Plans	SPECIFICATION CLARIFICATIONS
D1	X	X		Doc	Thermal sensor SMBus address latching
D2	X	X		Doc	MTRR initialization clarification
D3	X	X		Doc	Floating-point opcode clarification
D4	X	X		Doc	PI-ROM S-Spec clarification

Summary of Specification Changes

NO.	C0	C1	PKG	Plans	SPECIFICATION CHANGES
D1	X			Doc	Locks across cache line boundary disable bit added
D2	X	X		Doc	Non-GTL+ output leakage current change
D3	X	X		Doc	RESET# pin definition

ERRATA

D1. FP Data Operand Pointer May Be Incorrectly Calculated After FP Access Which Wraps 64-Kbyte Boundary in 16-Bit Code

Problem: The FP Data Operand Pointer is the effective address of the operand associated with the last noncontrol floating-point instruction executed by the machine. If an 80-bit floating-point access (load or store) occurs in a 16-bit mode other than protected mode (in which case the access will produce a segment limit violation), the memory access wraps a 64-Kbyte boundary, and the floating-point environment is subsequently saved, the value contained in the FP Data Operand Pointer may be incorrect.

Implication: A 32-bit operating system running 16-bit floating-point code may encounter this erratum, under the following conditions:

- The operating system is using a segment greater than 64 Kbytes in size.
- An application is running in a 16-bit mode other than protected mode.
- An 80-bit floating-point load or store which wraps the 64-Kbyte boundary is executed.
- The operating system performs a floating-point environment store (FSAVE/FNSAVE/FSTENV/FNSTENV) after the above memory access.
- The operating system uses the value contained in the FP Data Operand Pointer.

Wrapping an 80-bit floating-point load around a segment boundary in this way is not a normal programming practice. Intel has not currently identified any software which exhibits this behavior.

Workaround: If the FP Data Operand Pointer is used in an OS which may run 16-bit floating-point code, care must be taken to ensure that no 80-bit floating-point accesses are wrapped around a 64-Kbyte boundary.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D2. Differences Exist in Debug Exception Reporting

Problem: There exist some differences in the reporting of code and data breakpoint matches between that specified by previous Intel processors' specifications and the behavior of the Pentium II Xeon processor, as described below:

Case 1: The first case is for a breakpoint set on a MOVSS or POPSS instruction, when the instruction following it causes a debug register protection fault (DR7.gd is already set, enabling the fault). The processor reports delayed data breakpoint matches from the MOVSS or POPSS instructions by setting the matching DR6.bi bits, along with the debug register protection fault (DR6.bd). If additional breakpoint faults are matched during the call of the debug fault handler, the processor sets the breakpoint match bits (DR6.bi) to reflect the breakpoints matched by both the MOVSS or POPSS breakpoint and the debug fault handler call. The Pentium II Xeon processor only sets DR6.bd in both situation, and does not set any of the DR6.bi bits.

Case 2 In the second breakpoint reporting failure case, if a MOVSS or POPSS instruction with a data breakpoint is followed by a store to memory which:

a) crosses a 4-Kbyte page boundary,

OR

b) causes the page table Access or Dirty (A/D) bits to be modified,

the breakpoint information for the MOVSS or POPSS will be lost. Previous processors retain this information under these boundary conditions.

Case 3: If they occur after a MOVSS or POPSS instruction, the INT*n*, INTO, and INT3 instructions zero the DR6.Bi bits (bits B0 through B3), clearing pending breakpoint information, unlike previous processors.

Case 4: If a data breakpoint and an SMI (System Management Interrupt) occur simultaneously, the SMI will be serviced via a call to the SMM handler, and the pending breakpoint will be lost.

Case 5: When an instruction that accesses a debug register is executed, and a breakpoint is encountered on the instruction, the breakpoint is reported twice.

Implication When debugging or when developing debuggers for a Pentium II Xeon processor-based system, this behavior should be noted. Normal usage of the MOVSS or POPSS instructions (i.e., following them with a MOV ESP) will not exhibit the behavior of cases 1-3. Debugging in conjunction with SMM will be limited by case 4.

Workaround: Following MOVSS and POPSS instructions with a MOV ESP instruction when using breakpoints will avoid the first three cases of this erratum. No workaround has been identified for cases 4 or 5.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D3. FLUSH# Servicing Delayed While Waiting for STARTUP_IPI in MP Systems

Problem: In an MP system, if an application processor is waiting for a startup inter-processor interrupt (STARTUP_IPI), then it will not service a FLUSH# pin assertion until it has received the STARTUP_IPI.

Implication After the MP initialization protocol, only one processor becomes the bootstrap processor (BSP). The other processor becomes a slave application processor (AP). After losing the BSP arbitration, the AP goes into a wait loop, waiting for a STARTUP_IPI.

The BSP can wake up the AP to perform some tasks with a STARTUP_IPI, and then put it back to sleep with an initialization inter-processor interrupt (INIT_IPI, which has the same effect as asserting INIT#), which returns it to a wait loop. The result is a possible loss of cache coherency if the off-line processor is intended to service a FLUSH# assertion at this point. The FLUSH# will be serviced as soon as the processor is awakened by a STARTUP_IPI, before any other instructions are executed. Intel has not encountered any operating systems that are affected by this erratum.

Workaround: Operating system developers should take care to execute a WBINVD instruction before the AP is taken off-line using an INIT_IPI.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D4. Code Fetch Matching Disabled Debug Register May Cause Debug Exception

Problem: The bits L0-3 and G0-3 enable breakpoints local to a task and global to all tasks, respectively. If one of these bits is set, a breakpoint is enabled, corresponding to the addresses in the debug registers DR0-DR3. If at least one of these breakpoints is enabled, any of these registers are *disabled* (i.e., L*n* and G*n* are 0), and RW*n* for the disabled register is 00 (indicating a breakpoint on instruction execution), normally an instruction fetch will not cause an instruction-breakpoint fault based on a match with the address in the disabled register(s). However, if the address in a disabled register matches the address of a code fetch which also results in a page fault, an instruction-breakpoint fault will occur.

Implication: While debugging software, extraneous instruction-breakpoint faults may be encountered if breakpoint registers are not cleared when they are disabled. Debug software which does not implement a code breakpoint handler will fail, if this occurs. If a handler is present, the fault will be serviced. Mixing data and code may exacerbate this problem by allowing disabled data breakpoint registers to break on an instruction fetch.

Workaround: The debug handler should clear breakpoint registers before they become disabled.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D5. Double ECC Error on Read May Result in BINIT#

Problem For this erratum to occur, the following conditions must be met:

- Machine Check Exceptions (MCEs) must be enabled.
- A dataless transaction (such as a write invalidate) must be occurring simultaneously with a transaction which returns data (a normal read).
- The read data must contain a double-bit uncorrectable ECC error.

If these conditions are met, the Pentium II Xeon processor will not be able to determine which transaction was erroneous, and instead of generating an MCE, it will generate a BINIT#.

Implication: The bus will be reinitialized in this case. However, since a double-bit uncorrectable ECC error occurred on the read, the MCE handler (which is normally reached on a double-bit uncorrectable ECC error for a read) would most likely cause the same BINIT# event.

Workaround: Though the ability to drive BINIT# can be disabled in the Pentium II Xeon processor, which would prevent the effects of this erratum, overall system behavior would not improve, since the error which would normally cause a BINIT# would instead cause the machine to shut down. No other workaround has been identified.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D6. FP Inexact-Result Exception Flag May Not Be Set

Problem: When the result of a floating-point operation is not exactly representable in the destination format (1/3 in binary form, for example), an inexact-result (precision) exception occurs. When this occurs, the PE bit (bit 5 of the FPU status word) is normally set by the processor. Under certain rare conditions, this bit may not be set when this rounding occurs. However, other actions taken by the processor (invoking the software exception handler if the exception is unmasked) are not affected. This erratum can only occur if the floating-point operation which causes the precision exception is immediately followed by one of the following instructions:

- FST m32real
- FST m64real
- FSTP m32real
- FSTP m64real
- FSTP m80real
- FIST m16int
- FIST m32int
- FISTP m16int
- FISTP m32int
- FISTP m64int

Note that even if this combination of instructions is encountered, there is also a dependency on the internal pipelining and execution state of both instructions in the processor.

Implication: Inexact-result exceptions are commonly masked or ignored by applications, as it happens frequently, and produces a rounded result acceptable to most applications. The PE bit of the FPU status word may not always be set upon receiving an inexact-result exception. Thus, if these exceptions are unmasked, a floating-point error exception handler may not recognize that a precision exception occurred. Note that this is a “sticky” bit, i.e., once set by an inexact-result condition, it remains set until cleared by software.

Workaround: This condition can be avoided by inserting two NOP instructions between the two floating-point instructions.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D7. BTM for SMI Will Contain Incorrect FROM EIP

Problem: A system management interrupt (SMI) will produce a Branch Trace Message (BTM), if BTMs are enabled. However, the FROM EIP field of the BTM (used to determine the address of the instruction which was being executed when the SMI was serviced) will not have been updated for the SMI, so the field will report the same FROM EIP as the previous BTM.

Implication: A BTM which is issued for an SMI will not contain the correct FROM EIP, limiting the usefulness of BTMs for debugging software in conjunction with System Management Mode (SMM).

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D8. I/O Restart in SMM May Fail After Simultaneous MCE

Problem: If an I/O instruction (IN, INS, REP INS, OUT, OUTS, or REP OUTS) is being executed, and if the data for this instruction becomes corrupted, the Pentium II Xeon processor will signal a machine check exception (MCE). If the instruction is directed at a device which is powered down, the processor may also receive an assertion of SMI#. Since MCEs have higher priority, the processor will call the MCE handler, and the SMI# assertion will remain pending. However, upon attempting to execute the first instruction of the MCE handler, the SMI# will be recognized and the processor will attempt to execute the SMM handler. If the SMM handler is completed successfully, it will attempt to restart the I/O instruction, but will not have the correct machine state, due to the call to the MCE handler.

Implication: A simultaneous MCE and SMI# assertion may occur for one of the I/O instructions above. The SMM handler may attempt to restart such an I/O instruction, but will have corrupted state due to the MCE handler call, leading to failure of the restart and shutdown of the processor.

Workaround: If a system implementation must support both SMM and MCEs, the first thing the SMM handler code (when an I/O restart is to be performed) should do is check for a pending MCE. If there is an MCE pending, the SMM handler should immediately exit via an RSM instruction and allow the machine check exception handler to execute. If there is not, the SMM handler may proceed with its normal operation.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D9. Branch Traps Do Not Function if BTMs Are Also Enabled

Problem: If branch traps or branch trace messages (BTMs) are enabled alone, both function as expected. However, if both are enabled, only the BTMs will function, and the branch traps will be ignored.

Implication: The branch traps and branch trace message debugging features cannot be used together.

Workaround: If branch trap functionality is desired, BTMs must be disabled.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D10. Checker BIST Failure in FRC Mode Not Signaled

Problem: If a system is running in functional redundancy checking (FRC) mode, and the checker of the master-checker pair encounters a hard failure while running the built-in self test (BIST), the checker will tri-state all outputs without signaling an IERR#.

Implication: Assuming the master passes BIST successfully, it will continue execution unchecked, operating without functional redundancy. However, the necessary pull-up on the FRCERR pin will cause an FRCERR to be signaled. The operation of the master depends on the implementation of FRCERR.

Workaround: For successful detection of BIST failure in the checker of an FRC pair, use the FRCERR signal, instead of IERR#.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D11. BINIT# Assertion Causes FRCERR Assertion in FRC Mode

Problem: If a pair of Pentium II Xeon processors are running in functional redundancy checking (FRC) mode, and a catastrophic error condition causes BINIT# to be asserted, the checker in the master-checker pair will enter shutdown. The next bus transaction from the master will then result in the assertion of FRCERR.

Implication: Bus initialization via an assertion of BINIT# occurs as the result of a catastrophic error condition which precludes the continuing reliable execution of the system. Under normal circumstances, the master-checker pair would remain synchronized in the execution of the BINIT# handler. However, due to this erratum, an FRCERR will be signaled. System behavior then depends on the system specific error recovery mechanisms.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D12. Machine Check Exception Handler May Not Always Execute Successfully

Problem: An asynchronous machine check exception (MCE), such as a BINIT# event, which occurs during an access that splits a 4-Kbyte page boundary may leave some internal registers in an indeterminate state. Thus, MCE handler code may not always run successfully if an asynchronous MCE has occurred previously.

Implication: An MCE may not always result in the successful execution of the MCE handler. However, asynchronous MCEs usually occur upon detection of a catastrophic system condition that would also hang the processor. Leaving MCEs disabled will result in the condition which caused the asynchronous MCE instead causing the processor to enter shutdown. Therefore, leaving MCEs disabled may not improve overall system behavior.

Workaround: No workaround, which would guarantee successful MCE handler execution under this condition, has been identified.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D13. LBER May Be Corrupted After Some Events

Problem: The last branch record (LBR) and the last branch before exception record (LBER) can be used to determine the source and destination information for previous branches or exceptions. The LBR contains the source and destination addresses for the last branch or exception, and the LBER contains similar information for the last branch taken before the last exception. This information is typically used to determine the location of a branch which leads to execution of code which causes an exception. However, after a catastrophic bus condition which results in an assertion of BINIT# and the re-initialization of the buses, the value in the LBER may be corrupted. Also, after either a CALL which results in a fault or a software interrupt, the LBER and LBR will be updated to the same value, when the LBER should not have been updated.

Implication: The LBER and LBR registers are used only for debugging purposes. When this erratum occurs, the LBER will not contain reliable address information. The value of LBER should be used with caution when debugging branching code; if the values in the LBR and LBER are the same, then the LBER value is incorrect. Also, the value in the LBER should not be relied upon after a BINIT# event.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D14. BTMs May Be Corrupted During Simultaneous L1 Cache Line Replacement

Problem: When Branch Trace Messages (BTMs) are enabled and such a message is generated, the BTM may be corrupted when issued to the bus by the L1 cache if a new line of data is brought into the L1 data cache simultaneously. Though the new line being stored in the L1 cache is stored correctly, and no corruption occurs in the data, the information in the BTM may be incorrect due to the internal collision of the data line and the BTM.

Implication: Although BTMs may not be entirely reliable due to this erratum, the conditions necessary for this boundary condition to occur have only been exhibited during focused simulation testing. Intel has currently not observed this erratum in a system level validation environment.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D15. A20M# May Be Inverted After Returning from SMM and Reset

Problem: This erratum is seen when software causes the following events to occur:

1. The assertion of A20M# in real address mode.
2. After entering the 1-Mbyte address wrap-around mode caused by the assertion of A20M#, there is an assertion of SMI# intended to cause a Reset or remove power to the processor. Once in the SMM handler, software saves the SMM state save map to an area of nonvolatile memory from which it can be restored at some point in the future. Then software asserts RESET# or removes power to the processor.

After exiting Reset or completion of power-on, software asserts SMI# again. Once in the SMM handler, it then retrieves the old SMM state save map which was saved in event 2 above and copies it into the current SMM state save map. Software then asserts A20M# and executes the RSM instruction. After exiting the SMM handler, the polarity of A20M# is inverted.

Implication: If this erratum occurs, A20M# will behave with a polarity opposite from what is expected (i.e., the 1-Mbyte address wrap-around mode is enabled when A20M# is deasserted, and does not occur when A20M# is asserted).

Workaround: Software should save the A20M# signal state in nonvolatile memory before an assertion of RESET# or a power down condition. After coming out of Reset or at power on, SMI# should be asserted again. During the restoration of the old SMM state save map described in event 3 above, the entire map should be restored, except for bit 5 of the byte at offset 7F18h. This bit should retain the value assigned to it when the SMM state save map was created in event 3. The SMM handler should then restore the original value of the A20M# signal.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D16. Near CALL to ESP Creates Unexpected EIP Address

Problem: Pentium II processor-based systems may hang due to an internal protocol violation. When a snoopable transaction is issued on the bus and the cache line being accessed is in the modified state, the processor must deliver to the system bus an updated copy of the cache line. When the processor attempts to deliver the most up to date copy via an implicit writeback, the data transfer transaction fails and the DBSY# signal remains asserted until the next RESET#. This causes the system to hang indefinitely. In order to encounter this erratum, the following sequence of events must occur:

1. A snoopable transaction (transaction 1) is issued on the system bus. The processor contains in its L1 and/or L2 caches the data for this line in the modified state.
2. Another snoopable transaction (transaction 2) is issued and the processor contains this line only in its L2 cache in the modified state. Both of these transactions can be issued by either the chipset, by the processor (in which case they are of the self-snoop type), by another processor (2-way MP systems), or any combination thereof.
3. A nonsnoopable transaction is then issued (transaction 3) for which address bits A15-A5 are the same as those in transaction 2.
4. Transaction 3 is followed by a snoopable transaction (transaction 4).
5. The completion of the data transfer phase of transaction 1 must line up with the snoop response phase of transaction 3. This data transfer phase of transaction 1 must occur after the ADS# of transaction 4 and line up with the completion of an internal cache transaction.
6. The internal cache transaction must miss the L2 targeting a line for eviction, but the internal cache transaction must be such that it has to be retried.

The result of this sequence of transactions causes the processor bus to lock up after delivering the data for transaction 1, but prior to delivering the data for transaction 2. Since this data is never delivered, DBSY# does not deassert and the system hangs.

Implication: The Pentium II processor may cause a system to hang if the above listed sequence of events occurs. This sequence is a necessary condition to hit the erratum, but multiple variations of this sequence which also cause this erratum are also possible. The probability of encountering this erratum increases with I/O queue depth greater than 4 and in 2-way MP systems.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D17. Mixed Cacheability of Lock Variables Is Problematic in MP Systems

Problem: This errata only affects multiprocessor systems where a lock variable address is marked cacheable in one processor and uncacheable in any others. The processors which have it marked uncacheable may stall indefinitely when accessing the lock variable. The stall is only encountered if:

- One processor has the lock variable cached, and is attempting to execute a cache lock.
- If the processor which has that address cached has it cached in its L2 only.
- Other processors, meanwhile, issue back to back accesses to that same address on the bus.

Implication: MP systems where all processors either use cache locks or consistent locks to uncacheable space will not encounter this problem. If, however, a lock variable's cacheability varies in different processors, and several processors are all attempting to perform the lock simultaneously, an indefinite stall may be experienced by the processors which have it marked uncacheable in locking the variable (if the conditions above are satisfied). Intel has only encountered this problem in focus testing with artificially generated external events. Intel has not currently identified any commercial software which exhibits this problem.

Workaround: Follow a homogenous model for the memory type range registers (MTRRs), ensuring that all processors have the same cacheability attributes for each region of memory; do not use locks whose memory type is cacheable on one processor, and uncacheable on others. Avoid page table aliasing, which may produce a nonhomogenous memory model.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D18. MCE Due to L2 Parity Error Gives L1 MCACOD.LL

Problem: If a Cache Reply Parity (CRP) error, Cache Address Parity (CAP) error, or Cache Synchronous Error (CSER) occurs on an access to the Pentium II Xeon processor's L2 cache, the resulting Machine Check Architectural Error Code (MCACOD) will be logged with '01' in the LL field. This value indicates an L1 cache error; the value should be '10', indicating an L2 cache error. Note that L2 ECC errors have the correct value of '10' logged.

Implication: An L2 cache access error, other than an ECC error, will be improperly logged as an L1 cache error in MCACOD.LL.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D19. Memory Type Field Undefined for Nonmemory Operations

Problem: The Memory Type field for nonmemory transactions such as I/O and Special Cycles are undefined. Although the Memory Type attribute for nonmemory operations logically should (and usually does) manifest itself as UC, this feature is not designed into the implementation and is therefore inconsistent.

Implication: Bus agents may decode a non-UC memory type for nonmemory bus transactions.

Workaround: Bus agents must consider transaction type to determine the validity of the Memory Type field for a transaction.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D20. Infinite Snoop Stall During L2 Initialization of MP Systems

Problem: It is possible for snoop traffic generated on the system bus while a processor is executing its L2 cache initialization routine to cause the initializing processor to hang.

Implication: An MP system which does not suppress snoop traffic while L2 caches are being initialized may hang during this initialization sequence.

Workaround: System BIOS can create an execution environment which allows processors to initialize their L2 caches without the system generating any snoop traffic on the bus.
Below is a pseudo-code fragment, designed explicitly for a four-processor system, that uses a serial algorithm to initialize each processor's L2 cache:

```
Suppress_all_I/O_traffic()
 K = 0;
 while (K <= 3)
 {
 /* Obtain current value of K. This forces both Temp and K into */
 /* the L1 cache. Note that Temp could also be maintained in a */
 /* general purpose register.         */

 Temp = K;
 Wait_until_all_processors_are_signed_in_at_barrier()
  if ( logical_proc_APIC_id == K ) {
  {
    wait_10_usecs_delay_loop(); /* this time delay, required */
           /* in the worst case, allows */
           /* the barrier semaphore to */
           /* settle to shared state. */
    Initialize L2 cache
    K++
    }
  else
    while (Temp == K);
    }
  }
```

This algorithm prevents bus snoop traffic from the other processors, which would otherwise cause the initializing processor to hang. The algorithm assumes that the L1 cache is enabled (the Temp and K variables must be cached by each processor). Also, the Memory Type Range Register (MTRR) for the data segment must be set to WB (writeback) memory type.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D21. FP Data Operand Pointer May Not Be Zero After Power On or Reset

Problem: The FP Data Operand Pointer, as specified, should be reset to zero upon power on or Reset by the processor. Due to this erratum, the FP Data Operand Pointer may be nonzero after power on or Reset.

Implication: Software which uses the FP Data Operand Pointer and count on its value being zero after power on or Reset without first executing an FINIT/FNINIT instruction will use an incorrect value, resulting in incorrect behavior of the software.

Workaround: Software should follow the recommendation in Section 8.2 of the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide* (Order Number 243192). This recommendation states that if the FPU will be used, software-initialization code should execute an FINIT/FNINIT instruction following a hardware reset. This will correctly clear the FP Data Operand Pointer to zero.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D22. Premature Execution of Load Operation Prior to Exception Handler Invocation

Problem: This erratum can occur in any of the following situations:

1. If an instruction that performs a memory load causes a code segment limit violation,
2. If a waiting floating-point instruction or MMX™ instruction that performs a memory load has a floating-point exception pending, or
3. If an MMX instruction that performs a memory load and has either CR0.EM = 1 (Emulation bit set), a floating-point Top-of-Stack (FP TOS) not equal to 0, or a DNA exception pending.

If any of the above circumstances occur, it is possible that the load portion of the instruction will have executed before the exception handler is entered.

Implication: In normal code execution where the target of the load operation is to write back memory there is no impact from the load being prematurely executed, nor from the restart and subsequent re-execution of that instruction by the exception handler. If the target of the load is to uncached memory that has a system side-effect, restarting the instruction may cause unexpected system behavior due to the repetition of the side-effect.

Workaround: Code which performs loads from memory that has side-effects can effectively work around this behavior by using simple integer-based load instructions when accessing side-effect memory, and by ensuring that all code is written such that a code segment limit violation cannot occur as a part of reading from side-effect memory.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D23. EFLAGS Discrepancy on Page Fault After Multiprocessor TLB Shootdown

Problem: This erratum may occur when the Pentium II Xeon processor executes one of the following read-modify-write arithmetic instructions and a page fault occurs during the store of the memory operand: ADD, AND, BTC, BTR, BTS, CMPXCHG, DEC, INC, NEG, NOT, OR, ROL/ROR, SAL/SAR/SHL/SHR, SHLD, SHRD, SUB, XOR, and XADD. In this case, the EFLAGS value pushed onto the stack of the page fault handler may reflect the status of the register after the instruction would have completed execution rather than before it. The following conditions are required for the store to generate a page fault and call the operating system page fault handler:

1. The store address entry must be evicted from the DTLB by speculative loads from other instructions that hit the same way of the DTLB before the store has completed. DTLB eviction requires at least three load operations that have linear address bits 15:12 equal to each other and address bits 31:16 different from each other in close physical proximity to the arithmetic operation.
2. The page table entry for the store address must have its permissions tightened during the very small window of time between the DTLB eviction and execution of the store. Examples of page permission tightening include from Present to Not Present or from Read/Write to Read Only, etc.
3. Another processor, without corresponding synchronization and TLB flush, must cause the permission change.

Implication: This scenario may only occur on a multiprocessor platform running an operating system that performs “lazy” TLB shootdowns. The memory image of the EFLAGS register on the page fault handler’s stack prematurely contains the final arithmetic flag values although the instruction has not yet completed. Intel has not identified any operating systems that inspect the arithmetic portion of the EFLAGS register during a page fault nor observed this erratum in laboratory testing of software applications.

Workaround: No workaround is needed upon normal restart of the instruction, since this erratum is transparent to the faulting code and results in correct instruction behavior. Operating systems may ensure that no processor is currently accessing a page that is scheduled to have its page permissions tightened or have a page fault handler that ignores any incorrect state.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D24. Read Portion of RMW Instruction May Execute Twice

Problem: When the Pentium II Xeon processor executes a read-modify-write (RMW) arithmetic instruction, with memory as the destination, it is possible for a page fault to occur during the execution of the store on the memory operand after the read operation has completed but before the write operation completes. If the memory targeted for the instruction is UC (uncached), memory will observe the occurrence of the initial load before the page fault handler and again if the instruction is restarted.

Implication: If the memory targeted for the RMW instruction has no side effects, then the memory location will simply be read twice with no additional implications. If, however, the load targets a memory region that has side effects, multiple occurrences of the initial load may lead to unpredictable system behavior.

Workaround: Hardware and software developers who write device drivers for custom hardware that may have a side effect style of design should use simple loads and simple stores to transfer data to and from the device.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D25. Test Pin Must Be High During Power Up

Problem: The Pentium II Xeon processor core uses the PWRGOOD signal to ensure that no voltage sequencing issues arise; no pin assertions should cause the processor to change its behavior until this signal is asserted, when all power supplies and clocks to the processor are valid and stable. However, pin A23 of the Pentium II Xeon processor (TEST_VCC_CORE_A23), if at a low voltage level when the core power supply comes up, will cause the processor to enter an invalid test state.

Implication: If this erratum occurs, the system will fail to power up successfully.

Workaround: Ensure that this pin is pulled up using the core voltage supply. As this pin was previously named TEST_VCC_25_A23, some baseboards may need to tie this pin to 2.5 V. Such baseboards should use a 100 kΩ resistor to ensure proper sequencing on this pin (the internal pull-up will keep the signal from being asserted during power up). Alternatively, ensure that the 2.5 V power supply ($V_{CC2.5}$) reaches a valid level prior to the core voltage supply (V_{CCCORE}).

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D26. Processor Information ROM Uses WP Pin

Problem: The Pentium II Xeon processor contains two memory regions addressable by an SMBus master in a system. One is the Processor Information ROM, which contains Intel data as defined in the specification, and is intended to be write protected. The other is a Scratch EEPROM which may be defined by the OEM for system use. The Scratch EEPROM can be write-protected by asserting the WP pin on the Pentium II Xeon processor (pin B148). Deasserting this pin allows the Scratch EEPROM to be written using the SMBus. Due to this erratum, deasserting this pin will also allow the Processor Information ROM to be written if it is addressed instead of the Scratch EEPROM.

Implication: Care must be taken to address only the Scratch EEPROM when writing data using the SMBus. The Processor Information ROM may be overwritten with incorrect information due to this erratum.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D27. Intervening Writeback May Occur During Locked Transaction

Problem During a transaction which has the LOCK# signal asserted (i.e., a locked transaction), there is a potential for an explicit writeback caused by a previous transaction to complete while the bus is locked. The explicit writeback will only be issued by the processor which has locked the bus, and the lock signal will not be deasserted until the locked transaction completes, but the atomicity of a lock may be compromised by this erratum. Note that the explicit writeback is an expected cycle, and no memory ordering violations will occur. This erratum is, however, a violation of the bus lock protocol.

Implication: A chipset or third-party agent (TPA) which tracks bus transactions in such a way that locked transactions may only consist of a read-write or read-read-write-write locked sequence, with no transactions intervening, may lose synchronization of state due to the intervening explicit writeback. Systems using chipsets or TPAs which can accept the intervening transaction will not be affected.

Workaround: The bus tracking logic of all devices on the system bus should allow for the occurrence of an intervening transaction during a locked transaction.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D28. MC2_STATUS MSR Has Model-Specific Error Code and Machine Check Architecture Error Code Reversed

Problem: The *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide,* documents that for the MC*i*_STATUS MSR, bits 15:0 contain the MCA (machine-check architecture) error code field, and bits 31:16 contain the model-specific error code field. However, for the MC2_STATUS MSR, these bits have been reversed. For the MC2_STATUS MSR, bits 15:0 contain the model-specific error code field and bits 31:16 contain the MCA error code field.

Implication: A machine check error may be decoded incorrectly if this erratum on the MC2_STATUS MSR is not taken into account.

Workaround: When decoding the MC2_STATUS MSR, reverse the two error fields.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D29. Cache Access While Changing BBL_CR_CTL3 Configuration May Cause Hang

Problem: The Model Specific Register (MSR) at address 11E, named BBL_CR_CTL3, is used to configure the core-to-L2 cache interface in the Pentium II Xeon processor. If, during the cache configuration process, a write to BBL_CR_CTL3 occurs which changes the mode of operation of the Pentium II Xeon processor's L2 cache components, and a simultaneous access occurs to cacheable space (such as an L1 cache snoop), a hang condition may result.

Implication: If caching is enabled and cache snooping occurs during L2 configuration, the system may hang.

Workaround: It is possible for BIOS code to contain a workaround for this erratum.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D30. Thermal Sensor May Assert SMBALERT# Incorrectly

Problem: The Pentium II Xeon processor has a thermal sensor that monitors the processor core's temperature. The thermal sensor asserts SMBALERT# if the processor temperature exceeds the temperature limits set in the Alarm Threshold Registers (T_{HIGH}, T_{LOW}). It also sets the corresponding Status Register bits to identify the cause of the interrupt. Figure 1 gives one example of the how the SMBALERT# signal could be used in a system.

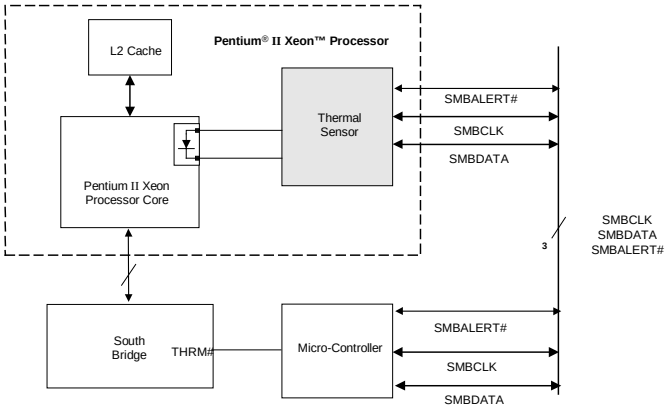


Figure 1. An Example of Microcontroller Driven Thermal Management

Under the conditions described below, the thermal sensor incorrectly generates the SMBALERT# interrupt. *All* of the following conditions must be met to trigger a false interrupt:

1. The thermal sensor must be in auto-convert mode.
2. The absolute value of the difference between the current temperature reading and the T_{HIGH} or T_{LOW} limit value must be less than or equal to 8° C.
3. The current temperature reading must be different from the previous reading.

With a false assertion of SMBALERT#, the corresponding bit in the Status Register (R_{HIGH}, and R_{LOW}) also will be incorrect.

Implication: There is no system impact from this erratum if temperature polling is used for processor thermal management. If the SMABLERT# interrupt is employed to manage processor thermal sensing, then servicing the false interrupt may result in premature system action depending on the software and hardware implementations used. The rate of the false interrupts is less than the auto-convert rate of the thermal sensor.

Workaround: Three different (mutually exclusive) workarounds are possible:

1. Before servicing an interrupt from the thermal sensor, read and compare the processor thermal reading with the threshold limits (T_{HIGH} or T_{LOW}). Figures 2 and 3 provide basic flowcharts for the implementation of this workaround in an interrupt driven system.
2. If the firmware implemented polls the Status Register only, then before taking any action, re-read the temperature register and do a comparison with the alarm threshold limits (T_{HIGH} or T_{LOW}) to determine if the value is actually still within the temperature window.
3. Use a temperature polling scheme to monitor the processor temperature

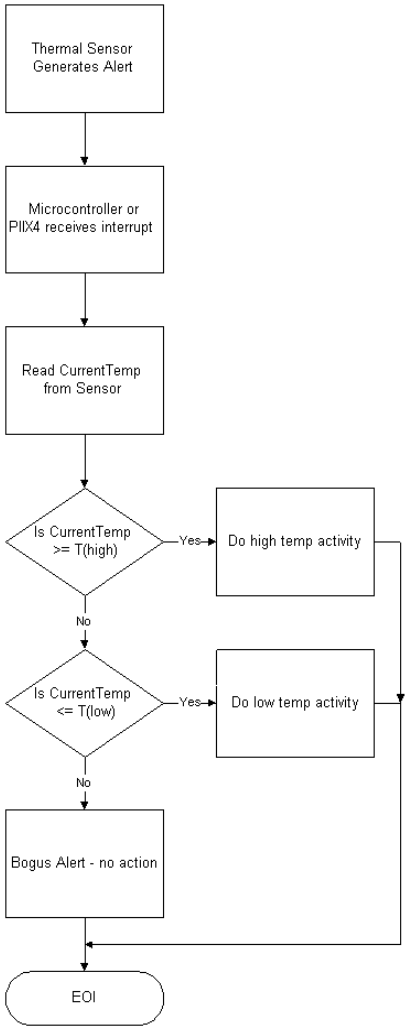


Figure 2. Workaround Flowchart: SMBALERT#-Driven System

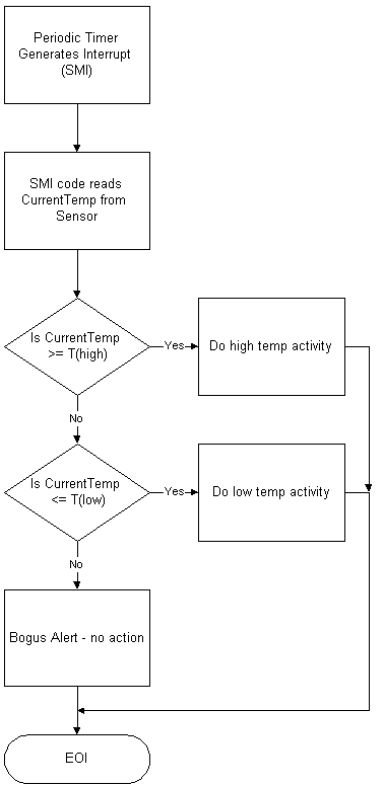


Figure 3. Workaround Flowchart: SMI#-Driven System

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D31. MOVD Following Zeroing Instruction Can Cause Incorrect Result

Problem: An incorrect result may be calculated after the following circumstances occur:

1. A register has been zeroed with either a SUB reg, reg instruction or an XOR reg, reg instruction,
2. A value is moved with sign extension into the same register's lower 16 bits; or a signed integer multiply is performed to the same register's lower 16 bits,
3. This register is then copied to an MMX™ technology register using the MOVD instruction prior to any other operations on the sign-extended value.

Specifically, the sign may be incorrectly extended into bits 16-31 of the MMX technology register. Only the MMX technology register is affected by this erratum.

The erratum only occurs when the 3 following steps occur in the order shown. The erratum may occur with up to 40 intervening instructions that do not modify the sign-extended value between steps 2 and 3.

1. XOR EAX, EAX
   or SUB EAX, EAX
2. MOVSX AX, BL
   or MOVSX AX, byte ptr <memory address> or MOVSX AX, BX
   or MOVSX AX, word ptr <memory address> or IMUL BL (AX implicit, opcode F6 /5)
   or IMUL byte ptr <memory address> (AX implicit, opcode F6 /5) or IMUL AX, BX (opcode 0F AF /r)
   or IMUL AX, word ptr <memory address> (opcode 0F AF /r) or IMUL AX, BX, 16 (opcode 6B /r ib)
   or IMUL AX, word ptr <memory address>, 16 (opcode 6B /r ib) or IMUL AX, 8 (opcode 6B /r ib)
   or IMUL AX, BX, 1024 (opcode 69 /r iw)
   or IMUL AX, word ptr <memory address>, 1024 (opcode 69 /r iw) or IMUL AX, 1024 (opcode 69 /r iw)
   or CBW
3. MOVD MM0, EAX

Note that the values for immediate byte/words are merely representative (i.e., 8, 16, 1024) and that any value in the range for the size is affected. Also, note that this erratum may occur with "EAX" replaced with any 32-bit general purpose register, and "AX" with the corresponding 16-bit version of that replacement. "BL" or "BX" can be replaced with any 8-bit or 16-bit general purpose register. The CBW and IMUL (opcode F6 /5) instructions are specific to the EAX register only.

In the example, EAX is forced to contain 0 by the XOR or SUB instructions. Since the four types of the MOVSX or IMUL instructions and the CBW instruction modify only bits 15:8 of EAX by sign extending the lower eight bits of EAX, bits 31:16 of EAX should always contain 0. This implies that when MOVD copies EAX to MM0, bits 31:16 of MM0 should also be 0. Under certain scenarios, bits 31:16 of MM0 are not 0, but are replicas of bit 15 (the 16th bit) of AX. This is noticeable when the value in AX after the MOVSX, IMUL or CBW instruction is negative, i.e., bit 15 of AX is a 1.

When AX is positive (bit 15 of AX is a 0), MOVD will always produce the correct answer. If AX is negative (bit 15 of AX is a 1), MOVD may produce the right answer or the wrong answer depending on the point in time when the MOVD instruction is executed in relation to the MOVSX, IMUL or CBW instruction.

Implication: The effect of incorrect execution will vary from unnoticeable, due to the code sequence discarding the incorrect bits, to an application failure. If the MMX technology-enabled application in which MOVD is used to manipulate pixels, it is possible for one or more pixels to exhibit the wrong color or position momentarily. It is also possible for a computational application that uses the MOVD instruction in the manner described above to produce incorrect data. Note that this data may cause an unexpected page fault or general protection fault.

Workaround: There are two possible workarounds for this erratum:

1. Rather than using the MOVSX-MOVD, IMUL-MOVD or CBW-MOVD pairing to handle one variable at a time, use the sign extension capabilities (PSRAW, etc.) within MMX technology for operating on multiple variables. This would result in higher performance as well.
2. Insert another operation that modifies or copies the sign-extended value between the MOVSX/IMUL/CBW instruction and the MOVD instruction as in the example below:

```
XOR EAX, EAX (or SUB EAX, EAX)
MOVSX AX, BL (or other MOVSX, other IMUL or CBW instruction)
*MOV EAX, EAX
MOVD MM0, EAX
```

*Note: MOV EAX, EAX is used here as it is fairly generic. Again, EAX can be any 32-bit register.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D32. Top 4 PAT Entries Not Usable With Mode B or Mode C Paging

Problem: The Page Attribute Table (PAT) contains eight entries, which must all be initialized and considered when setting up memory types for the Pentium II Xeon processor. However, in Mode B or Mode C paging, the top four entries do not function correctly for 4-Kbyte pages. Specifically, bit 7 of page table entries which translate addresses to 4-Kbyte pages should be used as the upper bit of a three-bit index to determine the PAT entry that specifies the memory type for the page. When Mode B (CR4.PSE = 1) and/or Mode C (CR4.PAE = 1) are enabled, the processor forces this bit to zero when determining the memory type, regardless of the value in the page table entry. The upper four entries of the PAT function correctly for 2-Mbyte and 4-Mbyte large pages (specified by bit 12 of the page directory entry for those translations).

Implication: Only the lower four PAT entries are useful for 4-Kbyte translations when Mode B or C paging is used. In Mode A paging (4-Kbyte pages only), all eight entries may be used. All eight entries may also be used for large pages in Mode B or C paging.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D33. MOV with Debug Register Causes Debug Exception

Problem: When in V86 mode, if a MOV instruction is executed on debug registers, a general-protection exception (#GP) should be generated, as documented in the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*, Section 14.2. However, in the case when the general detect enable flag (GD) bit is set, the observed behavior is that a debug exception (#DB) is generated instead.

Implication: With debug-register protection enabled (i.e., the GD bit set), when attempting to execute a MOV on debug registers in V86 mode, a debug exception will be generated instead of the expected general-protection fault.

Workaround: In general, operating systems do not set the GD bit when they are in V86 mode. The GD bit is generally set and used by debuggers. The debug exception handler should check that the exception did not occur in V86 mode before continuing. If the exception did occur in V86 mode, the exception may be directed to the general-protection exception handler.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D34. UC Write May Be Reordered Around a Cacheable Write

Problem: After a write occurs to a UC (uncacheable) region of memory, there exists a small window of opportunity where a subsequent write transaction targeted for a UC memory region may be reordered in front of a write targeted to a region of cacheable memory. This erratum can only occur during the following sequence of bus transactions:

- A write to memory mapped as UC occurs,
- A write to cacheable (WB or WT) memory which is in Shared or Invalid state in the L2 cache occurs, and
- During the bus snoop of the cacheable line, another store to UC memory occurs.

Implication: If this erratum occurs, the second UC write will be observed on the bus prior to the Bus Invalidate Line (BIL) or Bus Read Invalidate Line (BRIL) transaction for the cacheable write. This presents a small window of opportunity for a fast bus-mastering I/O device which triggers an action based on the second UC write to arbitrate and gain ownership of the bus prior to the completion of the cacheable write, possibly retrieving stale data.

Workaround: It is possible for BIOS code to contain a workaround for this erratum.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D35. Misprediction in Program Flow May Cause Unexpected Instruction Execution

Problem: To optimize performance through dynamic execution technology, the P6 architecture has the ability to predict program flow. In the event of a misprediction, the processor will normally clear the incorrect prediction, adjust the EIP to the correct location, and flush out any instructions it may have fetched from the misprediction. In circumstances where a branch misprediction occurs, the correct target of the branch has already been opportunistically fetched into the streaming buffers, and the L2 cycle caused by the evicted cache line is retried by the L2 cache, the processor may fail to flush out the retirement unit before the speculative program flow is committed to a permanent state.

Implication: The results of this erratum may range from no effect to unpredictable application or OS failure. Manifestations of this failure may result in:

- Unexpected values in EIP,
- Faults or traps (e.g., page faults) on instructions that do not normally cause faults,
- Faults in the middle of instructions, or
- Unexplained values in registers/memory at the correct EIP.

Workaround: It is possible for BIOS code to contain a workaround for this erratum.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D36. System Bus ECC May Report False Errors

Problem: The processor's ECC circuitry may fail to meet its frequency timing specification under certain environmental conditions. At the high end of the temperature specification and/or the low end of the voltage range, the processor may report false ECC errors.

Implication: If the system has data error checking enabled (bit [1] of the EBL_CR_POWERON register set to '1') and has Machine Check Architecture enabled, spurious double bit error detection can occur causing Machine Check Exceptions (MCEs) and spurious single bit errors to occur and be logged. Under some circumstances the processor may assert BINIT#, which in turn, may cause some systems to generate an MCE, and in others may cause a reboot.

Workaround: Disable system bus data error checking (set bit [1] of the EBL_CR_POWERON register to '0').

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D37. Full In Order Queue May Cause Infinite DBSY# Assertion

Problem: For this erratum to occur, there must be a high rate of code fetches from the core to its L2 cache, which must hit the L2 cache, AND in parallel an externally generated read transaction that hits a modified line FOLLOWED by 7 consecutive 0 length external transactions in rapid succession FOLLOWED by another external transaction that also hits a modified line.

Implication: The writeback data is not transferred to memory. No further bus transactions may be issued because the In-Order Queue is full.

Workaround: It is possible for BIOS code to contain a workaround for this erratum, which enables BIOS control of streaming buffers. Systems susceptible to this erratum should then disable streaming buffers.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D38. Data Breakpoint Exception in a Displacement Relative Near Call May Corrupt EIP

Problem: If a misaligned data breakpoint is programmed to the same cache line as the memory location where the stack push of a near call is performed and any data breakpoints are enabled, the processor will update the stack and ESP appropriately, but may skip the code at the destination of the call. Hence, program execution will continue with the next instruction immediately following the call, instead of the target of the call.

Implication: The failure mechanism for this erratum is that the call would not be taken; therefore, instructions in the called subroutine would not be executed. As a result, any code relying on the execution of the subroutine will behave unpredictably.

Workaround: Whether enabled or not, do not program a misaligned data breakpoint to the same cache line on the stack where the push for the near call is performed.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D39. Fault on REP CMPS/SCAS Operation May Cause Incorrect EIP

Problem: If either a General Protection Fault, Alignment Check Fault, or Machine Check Exception occur during the first iteration of a REP CMPS or a REP SCAS instruction, an incorrect EIP may be pushed onto the stack of the event handler if all the following conditions are true:

- The event occurs on the initial load performed by the instruction(s),
- The condition of the zero flag before the repeat instruction happens to be opposite of the repeat condition (i.e., REP/REPE/REPZ CMPS/SCAS with ZF = 0 or RENE/REPNZ CMPS/SCAS with ZF = 1), and
- The faulting micro-op and a particular micro-op of the REP instruction are retired in the retirement unit in a specific sequence.

The EIP will point to the instruction following the REP CMPS/SCAS instead of pointing to the faulting instruction.

Implication: The result of the incorrect EIP may range from no effect to unexpected application/OS behavior.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D40. System Bus ECC Not Functional With 2:1 Ratio

Problem: If a processor is underclocked at a core frequency to system bus frequency ratio of 2:1 and system bus ECC is enabled, the system bus ECC detection and correction will negatively affect internal timing dependencies.

Implication: If system bus ECC is enabled, and the processor is underclocked at a 2:1 ratio, the system may behave unpredictably due to these timing dependencies.

Workaround: All bus agents that support system bus ECC must disable it when a 2:1 ratio is used.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D41. RDMSR or WRMSR to Invalid MSR Address May Not Cause GP Fault

Problem: The RDMSR and WRMSR instructions allow reading or writing of MSRs (Model Specific Registers) based on the index number placed in ECX. The processor should reject access to any reserved or unimplemented MSRs by generating #GP(0). However, there are some invalid MSR addresses for which the processor will not generate #GP(0).

Implication: For RDMSR, undefined values will be read into EDX:EAX. For WRMSR, undefined processor behavior may result.

Workaround: Do not use invalid MSR addresses with RDMSR or WRMSR.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D42. Null Selectors May Cause FRC Errors in FRC-Enabled Systems

Problem: If a null selector (0000-0003h) is removed from the stack and placed into any data segment register (DS, ES, FS or GS), an undefined value may be put into the descriptor cache (the hidden portion of the segment register sometimes referred to as a "shadow register"; see "Segment Registers" in Chapter 3 of the *Intel Architecture Software Developer's Manual, Volume 3*). If this occurs in an FRC-enabled system, the master and checker processors may load different undefined values into their respective descriptor caches. If an IRET instruction occurs while these nonmatching undefined values are in the descriptor caches, an FRC (Functional Redundancy Checking) error will occur.

Implication: The FRCERR signal may be incorrectly asserted due to this erratum.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D43. SYSENTER/SYSEXIT Instructions Can Implicitly Load "Null Segment Selector" to SS and CS Registers

Problem: According to the processor specification, attempting to load a null segment selector into the CS and SS segment registers should generate a General Protection Fault (#GP). Although loading a null segment selector to the other segment registers is allowed, the processor will generate an exception when the segment register holding a null selector is used to access memory.

However, the SYSENTER instruction can implicitly load a null value to the SS segment selector. This can occur if the value in SYSENTER_CS_MSR is between FFF8h and FFFBh when the SYSENTER instruction is executed. This behavior is part of the SYSENTER/SYSEXIT instruction definition; the content of the SYSTEM_CS_MSR is always incremented by 8 before it is loaded into the SS. This operation will set the null bit in the segment selector if a null result is generated, but it does not generate a #GP on the SYSENTER instruction itself. An exception will be generated as expected when the SS register is used to access memory, however.

The SYSEXIT instruction will also exhibit this behavior for both CS and SS when executed with the value in SYSENTER_CS_MSR between FFF0h and FFF3h, or between FFE8h and FFEBh, inclusive.

Implication: These instructions are intended for operating system use. If this erratum occurs (and the OS does not ensure that the processor never has a null segment selector in the SS or CS segment registers), the processor's behavior may become unpredictable, possibly resulting in system failure.

Workaround: Do not initialize the SYSTEM_CS_MSR with the values between FFF8h and FFFBh, FFF0h and FFF3h, or FFE8h and FFEBh before executing SYSENTER or SYSEXIT.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D44. PRELOAD Followed by EXTEST Does Not Load Boundary Scan Data

Problem: According to the IEEE 1149.1 Standard, the EXTEST instruction would use data "typically loaded onto the latched parallel outputs of boundary-scan shift-register stages using the SAMPLE/PRELOAD instruction prior to the selection of the EXTEST instruction." As a result of this erratum, this method cannot be used to load the data onto the outputs.

Implication: Using the PRELOAD instruction prior to the EXTEST instruction will not produce expected data after the completion of EXTEST.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D45. Far Jump to New TSS With D-bit Cleared May Cause System Hang

Problem: A task switch may be performed by executing a far jump through a task gate or to a new Task State Segment (TSS) directly. Normally, when such a jump to a new TSS occurs, the D-bit (which indicates that the page referenced by a Page Table Entry (PTE) has been modified) for the PTE which maps the location of the previous TSS will already be set and the processor will operate as expected. However, if the D-bit is clear at the time of the jump to the new TSS, the processor will hang.

Implication: If an OS is used which can clear the D-bit for system pages, and which jumps to a new TSS on a task switch, then a condition may occur which results in a system hang. Intel has not identified any commercial software which may encounter this condition; this erratum was discovered in a focused testing environment.

Workaround: Ensure that OS code does not clear the D-bit for system pages (including any pages that contain a task gate or TSS). Use task gates rather than jumping to a new TSS when performing a task switch.

Status: For the steppings affected see the Summary Table of Changes at the beginning of this section.

D46. Illegal Opcode During L2 Cache Initialization

Problem: It is possible for the cache components in the 1-Mbyte and 2-Mbyte Pentium II Xeon processor to power up in a state such that they are not synchronized. During a read under these circumstances, the data in the cache is correct but the processor does not read the data correctly.

Implication: The processor may read invalid data after the cache is enabled during the Power-On Self Test (POST) phase of boot-up, most likely resulting in an invalid opcode being received by the processor, which would generate an invalid opcode exception.

Workaround: Intel recommends that the following BIOS instructions (or equivalent) be added to the Intel L2 Cache initialization module, just prior to enabling the L2 cache via BBL_CR_CTL3 [8]:

```
MOV         ECX, 11Eh             ; MSR (11Eh) is BBL_CR_CTL3
RDMSR                             ; read contents
PUSH        EAX                   ; save lower 32 bits
PUSH        EDX                   ; save upper 32 bits
AND         AL, 0E1h              ; isolate latency bits
OR          AL, 00Ah              ; set to a desktop latency value
WRMSR                             ; write new value out
POP         EDX                   ; restore original value determined
POP         EAX                   ; by the BIOS for latency
WRMSR                             ; write it back out
```

IMPORTANT NOTE

The above example code contains stack operations. If the BIOS L2 cache initialization code is executed in a pre-stack environment, the BIOS developer must ensure that the push/pop instruction pairs are replaced with another register save method. Also, the BIOS developer must ensure that the actual BIOS code does not corrupt existing code's register usage.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D47. Incorrect Chunk Ordering May Prevent Execution of the Machine Check Exception Handler After BINIT#

Problem: If a catastrophic bus error is detected which results in a BINIT# assertion, and the BINIT# assertion is propagated to the processor's L2 cache at the same time that data is being sent to the processor, then the data may become corrupted in the processor's L1 cache.

Implication: Since BINIT# assertion is due to a catastrophic event on the bus, the corrupted data will not be used. However, it may prevent the processor from executing the Machine Check Exception (MCE) handler, causing the system to hang.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D48. Resume Flag May Not Be Cleared After Debug Exception

Problem: The Resume Flag (RF) is normally cleared by the processor after executing an instruction which causes a debug exception (#DB). In the process of determining whether the RF needs to be cleared after executing the instruction, the processor uses an internal register containing stale data. The stale data may unpredictably prevent the processor from clearing the RF.

Implication: If this erratum occurs, further debug exceptions will be disabled.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D49. Thermal Sensor Leakage Current May Exceed Specification

Problem: The thermal sensor in the Pentium II Xeon processor cartridge violates the maximum input leakage current specification in Table 9 of the *Pentium® II Xeon™ Processor at 400 and 450 MHz* datasheet (10µA).

Implication: The thermal sensor incorporates input protection diodes on the SMBCLK and SMBDAT signals for ESD protection. The protection diodes can potentially clamp these lines to ~0.6 V when the $V_{CCSMBus}$ voltage supply to the cartridge is powered off. Hence, when $V_{CCSMBus}$ is powered down, the Pentium II Xeon processor thermal sensor may prevent SMBus transactions originating from SMBus devices powered by a different voltage source, but on the same SMBus, from occurring. SMBus devices that are powered from different voltage power planes are not typically located on the same SMBus. This erratum will not affect designs using this isolation technique.

Workaround: If it is desired to have SMBus devices powered by a source other than $V_{CCSMBus}$, these devices must be isolated from the Pentium II Xeon processor SMBus to ensure that this erratum does not occur.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D50. System Bus Address Parity Checking May Report False AERR#

Problem: The processor's address parity error detection circuit may fail to meet its frequency timing specification under certain environmental conditions. At the high end of the temperature specification and/or the low end of the voltage range, the processor may report false address parity errors.

Implication: If the system has AERR# drive enabled (bit [3] of the EBL_CR_POWERON resister set to '1') spurious address detection and reporting may occur. In some system configurations BINIT# may be asserted on the system bus. This may cause some systems to generate a machine check exception and in others may cause a reboot.

Workaround: Disable AERR# drive from the processor. AERR# drive may be disabled by clearing bit [3] in the EBL_CR_POWERON register. In addition, if the chipset allows, AERR# drive should be enabled from the chipset and AERR# observation enabled on the processor. AERR# observation on the processor is enabled by asserting A8# on the active-to-inactive transition of RESET#.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D51. Misaligned Locked Access to APIC Space Results in Hang

Problem: When the processor's APIC space is accessed with a misaligned locked access a machine check exception is expected. However, the processor's machine check architecture is unable to handle the misaligned locked access.

Implication: If this erratum occurs the processor will hang. Typical usage models for the APIC address space do not use locked accesses. This erratum will not affect systems using such a model.

Workaround: Ensure that all accesses to APIC space are aligned and/or not locked.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D52. Potential Loss of Data Coherency During MP Data Ownership Transfer

Problem: In MP systems, processors may be sharing data in different cache lines, referenced as line A and line B in the discussion below. When this erratum occurs (with the following example given for a 2-way MP system with processors noted as ‘P0’ and ‘P1’), P0 contains a shared copy of line B in its L1. P1 has a shared copy of Line A. Each processor must manage the necessary invalidation and snoop cycles before that processor can modify and source the results of any internal writes to the other processor.

There exists a narrow timing window when, if P1 requests a copy of line B it may be supplied by P0 in an Exclusive state which allows P1 to modify the contents of the line with no further external invalidation cycles. In this narrow window P0 may also retire instructions that use the original data present before P1 performed the modification.

Implication: Multiprocessor or threaded application synchronization, required for low level data sharing, that is implemented via operating system provided synchronization constructs are not affected by this erratum. Applications that rely upon the usage of locked semaphores rather than memory ordering are also unaffected. This erratum does not affect uniprocessor systems. The existence of this erratum was discovered during ongoing design reviews but it has not as yet been reproduced in a lab environment. Intel has not identified, to date, any commercially available application or operating system software which is affected by this erratum. If the erratum does occur one processor may execute software with the stale data that was present from the previous shared state rather than the data written more recently by another processor.

Workaround: Deterministic barriers beyond which program variables will not be modified can be achieved via the usage of locked semaphore operations. These should effectively prevent the occurrence of this erratum.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D53. Memory Ordering Based Synchronization May Cause a Livelock Condition in MP Systems

Problem: In an MP environment, the following sequence of code (or similar code) in two processors (P0 and P1) may cause them to each enter an infinite loop (livelock condition):

```
P0                                      P1
        MOV [xyz], EAX      (1)         wait1:  MOV EBX, [abc]      (2)
        .                                       CMP EBX, val1       (3)
        .                                       JNE wait1           (4)
        .
        MOV [abc], val1     (6)                 MOV [abc], val2     (5)
wait0:  MOV EBX, [abc]      (7)
                CMP EBX, val2       (8)
                JNE wait0           (9)
```

NOTE

EAX and EBX can be any general-purpose register. Addresses [abc] and [xyz] can be any location in memory and must be in the same bank of the L1 cache. Variables “val1” and “val2” can be any integer.

The algorithm above involves processors P0 and P1, each of which use loops to keep them synchronized with each other. P1 is looping until instruction (6) in P0 is globally observed. Likewise, P0 will loop until instruction (5) in P1 is globally observed.

The P6 architecture allows for instructions (1) and (7) in P0 to be dispatched to the L1 cache simultaneously. If the two instructions are accessing the same memory bank in the L1 cache, the load (7) will be given higher priority and will complete, blocking instruction (1).

Instructions (8) and (9) may then execute and retire, placing the instruction pointer back to instruction (7). This is due to the condition at the end of the “wait0” loop being false. The livelock scenario can occur if the timing of the wait0 loop execution is such that instruction (7) in P0 is ready for completion every time that instruction (1) tries to complete. Instruction (7) will again have higher priority, preventing the data ([xyz]) in instruction (1) from being written to the L1 cache. This causes instruction (6) in P0 to not complete and the sequence “wait0” to loop infinitely in P0.

A livelock condition also occurs in P1 because instruction (6) in P0 does not complete (blocked by instruction (1) not completing). The problem with this scenario is that P0 should eventually allow for instruction (1) to write its data to the L1 cache. If this occurs, the data in instruction (6) will be written to memory, allowing the conditions in both loops to be true.

Implication: Both processors will be stuck in an infinite loop, leading to a hang condition. Note that if P0 receives any interrupt, the loop timing will be disrupted such that the livelock will be broken. The system timer, a keystroke, or mouse movement can provide an interrupt that will break the livelock.

Workaround: Use a LOCK instruction to force P0 to execute instruction (6) before instruction (7).

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D54. GP# Fault on WRMSR to ROB_CR_BKUPTMPDR6

Problem: Writing a '1' to unimplemented bit(s) in the ROB_CR_BKUPTMPDR6 MSR (offset 1E0h) will result in a general protection fault (GP#).

Implication: The normal process used to write an MSR is to read the MSR using RDMSR, modify the bit(s) of interest, and then to write the MSR using WRMSR. Because of this erratum, this process may result in a GP# fault when used to modify the ROB_CR_BKUPTMPDR6 MSR.

Workaround: When writing to ROB_CR_BKUPTMPDR6 all unimplemented bits must be '0.' Implemented bits may be set as '0' or '1' as desired.

Status: For the steppings affected see the Summary of Changes at the beginning of this section.

D55. Machine Check Exception May Occur Due to Improper Line Eviction in the IFU

Problem: The Pentium II Xeon processor is designed to signal an unrecoverable Machine Check Exception (MCE) as a consistency checking mechanism. Under a complex set of circumstances involving multiple speculative branches and memory accesses there exists a one cycle long window in which the processor may signal a MCE in the Instruction Fetch Unit (IFU) because instructions previously decoded have been evicted from the IFU. The one cycle long window is opened when an opportunistic fetch receives a partial hit on a previously executed but not as yet completed store resident in the store buffer. The resulting partial hit erroneously causes the eviction of a line from the IFU at a time when the processor is expecting the line to still be present. If the MCE for this particular IFU event is disabled, execution will continue normally.

Implication: While this erratum may occur on a system with any number of Pentium II Xeon processors, the probability of occurrence increases with the number of processors. If this erratum does occur, a machine check exception will result. Note systems that implement an operating system that does not enable the Machine Check Architecture will be completely unaffected by this erratum (e.g., Windows* 95 and Windows 98).

Workaround: It is possible for BIOS code to contain a workaround for this erratum.

Status: For the steppings affected see the Summary of Changes at the beginning of this section.

D56. Lower Bits of SMRAM SMBASE Register Cannot Be Written With an ITP

Problem: The System Management Base (SMBASE) register (7EF8H) stores the starting address of the System Management RAM (SMRAM). This register is used by the processor when it is in System Management Mode (SMM), and its contents serve as the memory base for code execution and data storage. The 32-bit SMBASE register can normally be programmed to any value. When programmed with an In-Target Probe (ITP), however, any attempt to set the lower 11 bits of SMBASE to anything other than zeros via the WRMSR instruction will cause the attempted write to fail.

Implication: When set via an ITP, any attempt to relocate SMRAM space must be made with 2 Kbyte alignment.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D57. Task Switch Caused by Page Fault May Cause Wrong PTE and PDE Access Bit to be Set

Problem: If an operating system executes a task switch via a Task State Segment (TSS), and the TSS is wholly or partially located within a clean page (A and D bits clear) and the GDT entry for the new TSS is either misaligned across a cache line boundary or is in a clean page, the accessed and dirty bits for an incorrect page table/directory entry may be set.

Implication: An operating system that uses hardware task switching (or hardware task management) may encounter this erratum. The effect of the erratum depends on the alignment of the TSS and ranges from no anomalous behavior to unexpected errors.

Workaround: The operating system could align all TSSs to be within page boundaries and set the A and D bits for those pages to avoid this erratum. The operating system may alternately use software task management.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D58. Unsynchronized Cross-Modifying Code Operations Can Cause Unexpected Instruction Execution Results

Problem: The act of one processor, or system bus master, writing data into a currently executing code segment of a second processor with the intent of having the second processor execute that data as code is called cross-modifying code (XMC). XMC that does not force the second processor to execute a synchronizing instruction, prior to execution of the new code, is called unsynchronized XMC.

Software using unsynchronized XMC to modify the instruction byte stream of a processor can see unexpected instruction execution from the processor which is executing the modified code.

Implication: In this case, the phrase "unexpected execution behavior" encompasses the generation of most of the exceptions listed in the Intel Architecture Software Developer's Manual Volume 3: System Programming Guide including a General Protection Fault(GPF). In the event of a GPF the application executing the unsynchronized XMC operation would be terminated by the operating system.

Workaround: In order to avoid this erratum, programmers should use the XMC synchronization algorithm as detailed in the *Intel Architecture Software Developer's Manual Volume 3: System Programming Guide*, Section 7.1.3.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D59. Deadlock May Occur Due To Illegal-Instruction/Page-Miss Combination

Problem: Intel's 32-bit Instruction Set Architecture (ISA) utilizes most of the available op-code space; however, some byte combinations remain undefined and are considered illegal instructions. Intel processors detect the attempted execution of illegal instructions and signal an exception. This exception is handled by the operating system and/or application software.

Under a complex set of internal and external conditions involving illegal instructions, a deadlock may occur within the processor. The necessary conditions for the deadlock are:

1. An illegal instruction is executed.
2. Two page table walks occur within a narrow timing window coincident with the illegal instruction.

Implication: The illegal instructions involved in this erratum are unusual and invalid byte combinations that are not useful to application software or operating systems. These combinations are not normally generated in the course of software programming, nor are such sequences known by Intel to be generated in commercially available software and tools. Development tools (compilers, assemblers) do not generate this type of code sequence, and will normally flag such a sequence as an error. If this erratum occurs, the processor deadlock condition will occur, resulting in a system hang. Code execution cannot continue without a system RESET.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D60. FLUSH# Assertion Following STPCLK# May Prevent CPU Clocks From Stopping

Problem: If FLUSH# is asserted after STPCLK# is asserted, the cache flush operation will not occur until after STPCLK# is de-asserted. Furthermore, the pending flush will prevent the processor from entering the Sleep state, since the flush operation must complete prior to the processor entering the Sleep state.

Implication: Following SLP# assertion, processor power dissipation may be higher than expected.

Workaround: For systems that use the FLUSH# input signal and Deep Sleep state of the processor, ensure that FLUSH# is not asserted while STPCLK# is asserted.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D61. Floating-Point Exception Condition May be Deferred

Problem: If an operating system executes a task switch via a Task State Segment (TSS), and the TSS is wholly or partially located within a clean page (A and D bits clear) and the GDT entry for the new TSS is either misaligned across a cache line boundary or is in a clean page, the accessed and dirty bits for an incorrect page table/directory entry may be set.

Implication: An operating system that uses hardware task switching (or hardware task management) may encounter this erratum. The effect of the erratum depends on the alignment of the TSS and ranges from no anomalous behavior to unexpected errors.

Workaround: The operating system could align all TSSs to be within page boundaries and set the A and D bits for those pages to avoid this erratum. The operating system may alternately use software task management.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D62. Race Conditions May Exist on Thermal Sensor SMBus Collision Detection/Arbitration Circuitry

Problem: In certain SMBus configurations when the thermal sensor is used in “hard wired alert” mode along with at least one other device on the bus, the thermal sensor may continue to send its address after losing a collision arbitration in response to an Alert Response Address (ARA) by the SMBus controller.

In order for this erratum to occur, all of the following conditions must be present:

1. The thermal sensor must be configured with the alert enabled (default setting).
2. There must be one or more other devices on the SMBus along with the thermal sensor.
3. One or more of these other devices must be also configured with the alert enabled.
4. One or more of these other devices must have a lower address (higher priority) than the thermal sensor.
5. The thermal sensor must generate an SM alert while at least one other device has an SM alert pending to be serviced.

In this situation, the thermal sensor will continue to send its address on the SMBus even if it has a lower priority than the pending alert. When this occurs, the SMBus controller cannot correctly interpret the device address. This may cause the thermal sensor’s alert flag not to clear and may result in SMBus lockup.

Implication: The SMBus controller may see an invalid address and the resulting response of the SMBus controller will vary from implementation to implementation.

Workaround: Remove any of the five conditions listed above or:

1. In software, use polling mode for the thermal sensor data collection with the alert disabled. This software workaround has been validated on both Intel’s test platforms as well as on certain OEM systems.
2. Ensure that the thermal sensor alert may be cleared by a hardware or software mechanism. The implementation of this workaround will be system dependent.

Status: For the steppings affected, see the *Summary of Changes* at the beginning of this section.

D63. Selector for the LTR/LLDT register may get corrupted.

Problem: The internal selector portion of the respective register (TR, LDTR) may get corrupted if, during a small window of LTR or LLDT system instruction execution, the following sequence of events occur:

1. Speculative write to a segment register that might follow the LTR or LLDT instruction

2. The read segment descriptor of LTR/LLDT operation spans a page (4 Kbytes) boundary; or causes a page fault

Implication: Incorrect selector for LTR, LLDT instruction could be used after a task switch.

Workaround: Software can insert a serializing instruction between the LTR or LLDT instruction and the segment register write.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D64. INIT does not clear global entries in TLB.

Problem: INIT may not flush a TLB entry when:

1. The processor is in protected mode with paging enabled and the page global enable flag is set (PGE bit of CR4 register);
2. G bit for the page table entry is set;
3. TLB entry is present in TLB when INIT occurs.

Implication: Software may encounter unexpected page fault or incorrect address translation due to a TLB entry erroneously left in TLB after INIT.

Workaround: Write to CR3, CR4 or CR0 registers before writing to memory early in BIOS code to clear all the global entries from TLB.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D65. VM Bit Will Be Cleared on a Double Fault Handler

Problem: Following a task switch to a Double Fault Handler that was initiated while the processor was in virtual-8086 (VM86) mode, the VM bit will be incorrectly cleared in EFLAGS.

Implication: When the OS recovers from the double fault handler, the processor will no longer be in VM86 mode.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D66. Memory Aliasing with Inconsistent A and D Bits May Cause Processor Deadlock

Problem: In the event that software implements memory aliasing by having two Page Directory Entries (PDEs) point to a common Page Table Entry(PTE) and the accessed and dirty bits for the two PDEs are allowed to become inconsistent the processor may become deadlocked.

Implication: This erratum has not been observed with commercially available software.

Workaround: Software that needs to implement memory aliasing in this way should manage the consistency of the accessed and dirty bits

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D67. Use of Memory Aliasing with Inconsistent Memory Type May Cause System Hang

Problem: Software that implements memory aliasing by having more than one linear address mapped to the same physical page with different cache types may cause the system to hang. This would occur if one of the addresses is non-cacheable used in code segment and the other a cacheable address. If the cacheable address finds its way in instruction cache, and non-cacheable address is fetched in IFU, the processor may invalidate the non-cacheable address from the fetch unit. Any micro-architectural event that causes instruction restart will expect this instruction to still be in fetch unit and lack of it will cause system hang.

Implication: This erratum has not been observed with commercially available software.

Workaround: Although it is possible to have a single physical page mapped by two different linear addresses with different memory types, Intel has strongly discouraged this practice as it may lead to undefined results. Software that needs to implement memory aliasing should manage the memory type consistency.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D68. Processor may Report Invalid TSS Fault Instead of Double Fault During Mode-C Paging

Problem: When an operating system executes a task switch via a Task State Segment (TSS) the CR3 register is always updated from the new task TSS. In the mode C paging, once the CR3 is changed the processor will attempt to load the PDPTRs. If the CR3 from the target task TSS or task switch handler TSS is not valid then the new PDPTR will not be loaded. This will lead to the reporting of invalid TSS fault instead of the expected double fault.

Implication: Operating systems that access an invalid TSS may get invalid TSS fault instead of a double fault.

Workaround: Software needs to ensure any accessed TSS is valid.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D69. Machine Check Exception May Occur When Interleaving Code Between Different Memory Types

Problem: A small window of opportunity exists where code fetches interleaved between different memory types may cause a machine check exception. A complex set of micro-architectural boundary conditions is required to expose this window.

Implication: Interleaved instruction fetches between different memory types may result in a machine check exception. The system may hang if machine check exceptions are disabled. Intel has not observed the occurrence of this erratum while running commercially available applications or operating systems.

Workaround: Software can avoid this erratum by placing a serializing instruction between code fetches which span different memory types.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D1AP. APIC Access to Cacheable Memory Causes Shutdown

Problem: APIC operations that access memory with any type other than uncacheable (UC) are illegal. If an APIC operation to a memory type other than UC occurs and Machine Check Exceptions (MCEs) are disabled, the processor will enter shutdown after such an access. If MCEs are enabled, an MCE will occur. However, in this circumstance, a second MCE will be signaled. The second MCE signal will cause the Pentium II Xeon processor to enter shutdown.

Implication: Recovery from a PIC access to cacheable memory will not be successful. Software that accesses only UC type memory during APIC operations will not encounter this erratum.

Workaround: Ensure that the memory space to which PIC accesses can be made is marked as type UC (uncacheable) in the memory type range registers (MTRRs) to avoid this erratum.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

D2AP. MP Systems May Hang Due to Catastrophic Errors During BSP Determination

Problem: In MP systems, a catastrophic error during the bootstrap processor (BSP) determination process should cause the assertion of IERR#. If the catastrophic error is due to the APIC data bus being stuck at electrical zero, then the system hangs without asserting IERR#.

Implication: MP systems may hang during boot due to a catastrophic error. This erratum has not been observed to date in a typical commercial system, but was found during focused system testing using a grounded APIC data bus.

Workaround: None identified

Status: For the steppings affected see the Summary of Changes at the beginning of this section.

D3AP. Write to Mask LVT (Programmed as EXTINT) Will Not Deassert Outstanding Interrupt

Problem: If the APIC subsystem is configured in Virtual Wire Mode implemented through the local APIC, (i.e., the 8259 INTR signal is connected to LINT0 and LVT1's interrupt delivery mode field is programmed as EXTINT), a write to LVT1 intended to mask interrupts will not deassert the internal interrupt source if the external LINT0 signal is already asserted. The interrupt will be erroneously posted to the Pentium II Xeon processor despite the attempt to mask it via the LVT.

Implication: Because of the masking attempt, interrupts may be generated when the system software expects no interrupts to be posted.

Workaround: Software can issue a write to the 8259A interrupt mask register to deassert the LINT0 interrupt level, followed by a read to the controller to ensure that the LINT0 signal has been deasserted. Once this is ensured, software may then issue the write to mask LVT entry 1.

Status: For the steppings affected see the Summary of Changes at the beginning of this section.

DOCUMENTATION CHANGES

The Documentation Changes listed in this section apply to the following documents:

- *Pentium® II Xeon™ Processor at 400 and 450 MHz* datasheet
- *P6 Family of Processors Hardware Developer's Manual*
- *Intel Architecture Software Developer's Manual*, *Volumes 1, 2, and 3*

All Documentation Changes will be incorporated into a future version of the appropriate Pentium II Xeon processor documentation.

D1. STPCLK# Pin Definition

The *P6 Family of Processors Hardware Developer's Manual* and the *Pentium® II Xeon™ Processor at 400 and 450 MHz* datasheet both have incorrect definitions of the STPCLK# pin in their alphabetical signal listing. These documents incorrectly state:

> The processor continues to snoop bus transactions and *service interrupts* while in Stop Grant state. When STPCLK# is deasserted, the processor restarts its internal clock to all units and resumes execution.

They should state:

> The processor continues to snoop bus transactions and *may latch interrupts* while in Stop Grant state. When STPCLK# is deasserted, the processor restarts its internal clock to all units, resumes execution, *and services any pending interrupts*.

D2. Invalidating Caches and TLBs

Section 2.6.4 of the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide* incorrectly states:

> The INVD (invalidate cache with no writeback) instruction invalidates all data and instruction entries in the internal caches and TLBs and sends a signal to the external caches indicating that they should be invalidated also.

It should state:

> The INVD (invalidate cache with no writeback) instruction invalidates all data and instruction entries in the internal caches and sends a signal to the external caches indicating that they should be invalidated also.

D3. Handling of Self-Modifying and Cross-Modifying Code

Section 7.1.3 paragraph 1. of the *Intel Architecture Software Developer's Manual Vol 3: System Programming* incorrectly states:

> The act of a processor writing data into a currently executing code segment with the intent of executing that data as code is called **self-modifying code**. Intel Architecture processors exhibit model-specific behavior when executing self-modified code, depending upon how far ahead of the current execution pointer the code has been modified. As processor architectures become more complex and start to speculatively execute code ahead of the retirement point (as in the P6 family processors), the rules regarding which code should execute, pre- or post-modification, become blurred. To write self-modifying code and ensure that it is compliant with current and future Intel Architectures one of the following two coding options **should** be chosen.

It should state:

> The act of a processor writing data into a currently executing code segment with the intent of executing that data as code is called **self-modifying code**. Intel Architecture processors exhibit model-specific behavior when executing self-modified code, depending upon how far ahead of the current execution pointer the code has been modified. As processor architectures become more complex and start to speculatively execute code ahead of the retirement point (as in the P6 family processors), the rules regarding which code should execute, pre- or post-modification, become blurred. To write self-modifying code and ensure that it is compliant with current and future Intel Architectures one of the following two coding options **must** be chosen.

Section 7.1.3 paragraph 6 of the *Intel Architecture Software Developer's Manual Vol 3: System Programming* incorrectly states:

> The act of one processor writing data into the currently executing code segment of a second processor with the intent of having the second processor execute that data as code is called **cross-modifying code**. As with self-modifying code, Intel Architecture processors exhibit model-specific behavior when executing cross-modifying code, depending upon how far ahead of the executing processors current execution pointer the code has been modified. To write cross-modifying code and insure that it is compliant with current and future Intel Architectures, the following processor synchronization algorithm **should** be implemented.

It should state:

> The act of one processor writing data into the currently executing code segment of a second processor with the intent of having the second processor execute that data as code is called **cross-modifying code.** As with self-modifying code, Intel Architecture processors exhibit model-specific behavior when executing cross-modifying code, depending upon how far ahead of the executing processors current execution pointer the code has been modified. To write cross-modifying code and insure that it is compliant with current and future Intel Architectures, the following processor synchronization algorithm **must** be implemented.

D4. Machine Check Architecture Initialization of MCi_STATUS Registers

Section 12.5, the last paragraph of the *Intel Architecture Software Developer's Manual Vol. 3: System Programming Guide* incorrectly states:

> The processor can write valid information (such as an ECC error) into the MCi_STATUS registers while it is being powered up. As part of the initialization of the MCE exception handler, software might examine all the MCi_STATUS registers and log the contents of them, then rewrite them all to zeroes. This procedure is not included in the initialization pseudocode in Example 12-1.

It should state:

> The processor can write valid information (such as an ECC error) into the MCi_STATUS registers while it is being powered up. As part of the initialization of the MCE exception handler, software might examine all the MCi_STATUS registers and log the contents of them, then rewrite them all to zeroes. Following power cycling, the MCi_STATUS registers are not guaranteed to have valid data until after the registers are initially cleared to all zeroes by software. This procedure is not included in the initialization pseudocode in Example 12-1.

D5. LOCK# Signal Prefix Operands

Page 3-273, the first sentence of the third paragraph of the *Intel Architecture Software Developer's Manual Volume 2: Instruction Set Reference* states:

> The LOCK prefix can be prepended only to the following instructions and to those forms of the instructions that use a memory operand: ADD, ADC, AND, …

It should state:

> The LOCK prefix can be prepended only to the following instructions and to those forms of the instructions that use a memory operand as the destination operand only: ADD, ADC, AND, ...
>
> If the LOCK prefix is used with the memory operand as the source operand then an Invalid Opcode (Undefined Opcode), #UD, can occur.

D6. SMRAM State Save Map Contains Documentation Errors

In the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*, revision 2 Chapter 12, "System Management Mode (SMM)," Table 12-1 incorrectly documents the SMBASE+Offset for LDT Base on the P6 family of processors.

> The storage locations for these parameters are model specific (i.e., they may differ between the Pentium® processor, the Pentium® Pro processor, and other P6 family proliferations). **These entries in the tables above will be changed to Reserved. Hardware and software may not rely on the contents of these Reserved regions.**

D7. System Management Interrupt (SMI) During Startup IPI Clarification

The note on section 12.2 of Intel Architecture Software Developer's Manual Vol. 3: System Management Interrupt (SMI) states:

> In the P6 families of processors, when a processor that is designated as the application processor during an MP initialization protocol is waiting for a startup IPI (SIPI), it is in a mode where SMIs are masked.

It should state:

> In the P6 family of processors, when a processor that is designated as the application processor during an MP initialization protocol is waiting for a startup IPI (SIPI), it is in a mode where SMIs are masked. However, if SMI is received while an application processor is in the wait for SIPI mode, it will be pended. The processor will respond on receipt of a SIPI by immediately servicing the pended SMI and will go into SMM before executing from the SIPI vector.

D8. Memory Aliasing with Different Memory Types

The 4th paragraph of section 9.13.5 of Intel Architecture Software Developer's Manual Vol. 3: Programming the PAT states:

> The PAT allows any memory type to be specified in the page table, and therefore it is possible to have a single physical page mapped to two different linear addresses with different memory types. This practice is strongly discouraged by Intel and should be avoided as it may lead to undefined results.

It should change to:

> The PAT allows any memory type to be specified in the page table, and therefore it is possible to have a single physical page mapped to two different linear addresses with different memory types. Intel does not support this practice as it may lead to undefined operations including processor hang.

D9. RUNBIST Will Not Function When STPCLK# Driven Low

Paragraph 5 of Section 6.3 in the *P6 Family of Processors Hardware Developer's Manual* currently states:

> Note that RUNBIST will not function when the processor core clock has been stopped. All other 1149.1-defined instructions operate independently of the processor core clock.

It should state:

> Note that RUNBIST will not function when the processor STPCLK# input has been driven low. All other 1149.1-defined instructions will correctly operate regardless of the STPCLK# signal state.

D10. Memory Aliasing with Inconsistent A and D Bits may Cause Processor Deadlock

Add the following note in Chapter 3 and Chapter 9.13.5 of the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*, Revision 2:

> Processor allows memory aliasing by having two Page Directory Entries (PDEs) point to a common Page Table Entry (PTE). Software that needs to implement memory aliasing in this way should manage the consistency of the Accessed and Dirty bits. Allowing the Accessed and Dirty bits for the two PDEs to become inconsistent may lead to a processor deadlock.

D11. An Interrupt Could Occur While TSS is Marked Busy

Paragraph 5 of section 6.1.3 in the Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide, currently states:

> For all Intel Architecture processors, tasks are not recursive. A task cannot call or jump to itself.

It should state:

> For all Intel Architecture processors, tasks are not recursive. A task cannot call or jump to itself. Because Intel Architecture tasks are not re-entrant, a task used as an interrupt handler must have interrupts disabled between the time it completes the interrupt and the time it exits with IRET. Otherwise, another interrupt could occur while the interrupt task's TSS is still marked busy, causing a #GP fault.

D12. NMI Unmasked Early When Processor is Running in V86 Mode

Section 5.5.1 in the Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide, currently states:

> While an NMI interrupt handler is executing, the processor disables additional calls to the NMI handler until the next IRET instruction is executed. This blocking of subsequent NMIs prevents stacking up calls to NMI handler. It is recommended that the NMI interrupt handler be accessed through an interrupt gate to disable maskable hardware interrupts (refer to section 5.6.1, "Masking Maskable Hardware Interrupts").

It should state:

> While an NMI interrupt handler is executing, the processor disables additional calls to the NMI handler until the next IRET instruction is executed. This blocking of subsequent NMIs prevents stacking up calls to NMI handler. It is recommended that the NMI interrupt handler be accessed through an interrupt gate to disable maskable hardware interrupts (refer to section 5.6.1, "Masking Maskable Hardware Interrupts"). If the NMI handler is a virtual-8086 task with IOPL less than 3, the IRET from this handler triggers a general-protection exception (#GP) (section 16.2.7). In this case, NMI is unmasked before the #GP handler is invoked.

D13. P6 Reads Two Bytes for POP SEG Instruction

Paragraph 1 and 2 of section 18.24.1 in the Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide, currently states:

> When pushing a segment selector onto the stack, the Intel486™ processor writes 2 bytes onto 4-byte stacks and decrements ESP by 4. The P6 family and Pentium® processors write 4 bytes, with the upper 2 bytes being zeros.
>
> When popping a segment selector from the stack, the Intel486™ processor reads only 2 bytes. The P6 family and Pentium® processors read 4 bytes and discard the upper 2 bytes. This operation may have an effect if the ESP is close to the stack-segment limit. On the P6 family and Pentium® processors, stack location at ESP plus 4 may be above the stack limit, in which case a stack fault exception (#SS) will be generated. On the Intel486™ processor, stack location at ESP plus 2 may be less than the stack limit and no exception is generated.

It should state:

> When pushing a segment selector, Intel486™ and P6 family processors decrement ESP by the operand size and then write two bytes. If the operand size is 32-bits, the upper two bytes of the write are unmodified. The Intel Pentium® processor decrements ESP by the operand size and determines the size of the write by the operand size. If the operand size is 32-bits, the upper two bytes of the write are zero.
>
> When popping a segment selector, Intel486™ and P6 family processors read two bytes and increment ESP by the operand size of the instruction. The Intel Pentium® processor determines the size of the read by the operand size and increments ESP by the operand size.
>
> It is possible to align a 32-bit selector push or pop such that the operation will generate an exception on the Intel Pentium® processor and not on the Intel486™ and P6 family processors. This could occur if the third and/or fourth byte of the operation lies beyond the limit of the segment or if the third and/or fourth byte of the operation is located on a not-present or inaccessible page.

D14. APIC Register Offsets are Aligned on 128-bit Boundaries

Paragraph 3 of section 7.5.7 in the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*, currently states:

> Within the 4KB APIC register area, the register address allocation scheme is shown in Table 7-1. Register offsets are aligned on 128-bit boundaries. All registers must be accessed using 32-bit loads and stores. Wider registers (64-bit or 256-bit) are defined and accessed as independent multiple 32-bit registers. If a LOCK prefix is used with a MOV instruction that accesses the APIC address space, the prefix is ignored; that is a locking operation does not take place.

It should say:

> Within the 4KB APIC register area, the register address allocation scheme is shown in Table 7-1. Register offsets are aligned on 128-bit boundaries. All registers must be accessed using loads and stores that are aligned on 128-bit boundaries and manipulate the registers as 32-bit quantities. Wider registers (64-bit or 256-bit) are defined and accessed as independent multiple 32-bit registers. If a LOCK prefix is used with a MOV instruction that accesses the APIC address space, the prefix is ignored; that is a locking operation does not take place.

D15. Single Stepping of Instructions Breaks out of HALT State

Paragraph 1 of section 2.6.5 in the Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide, currently states:

> The HLT (halt processor) instruction stops the processor until an enabled interrupt (such as NMI or SMI, which are normally enabled), the BINIT# signal, the INIT# signal, or the RESET# signal is received. The processor generates a special bus cycle to indicate that the halt mode has been entered. Hardware may respond to this signal in a number of ways. An indicator light on the front panel may be turned on. An NMI interrupt for recording diagnostic information may be generated. Reset initialization may be invoked. (Note that the BINIT# pin was introduced with the Pentium® Pro processor.)

It should say:

> The HLT (halt processor) instruction stops the processor until an enabled interrupt (such as NMI, or SMI which are normally enabled), a debug exception, the BINIT# signal, the INIT# signal, or the RESET# signal is received. The processor generates a special bus cycle to indicate that the halt mode has been entered. Hardware may respond to this signal in a number of ways. An indicator light on the front panel may be turned on. An NMI interrupt for recording diagnostic information may be generated. Reset initialization may be invoked. (Note that the BINIT# pin was introduced with the Pentium® Pro processor)

D16. Additional Signal Resumes Execution While in a HALT State

Paragraph 1 of the Description section on page 3-291 in the Intel Architecture Software Developer's Manual, Volume 2: Instruction Set Reference Manual, currently states:

> This instruction stops instruction execution and places the processor in a HALT state. An enabled interrupt, NMI, or a reset will resume execution. If an interrupt (including NMI) is used to resume execution after a HLT instruction, the saved instruction pointer (CS:EIP) points to the instruction following the HLT instruction.

It should say:

> This instruction stops instruction execution and places the processor in a HALT state. An enabled interrupt (including NMI and SMI), a debug exception, the BINIT# signal, the INIT# signal, or the RESET# signal will resume execution. If an interrupt (including NMI) is used to resume execution after a HLT instruction, the saved instruction pointer (CS:EIP) points to the instruction following the HLT instruction.

SPECIFICATION CLARIFICATIONS

The Specification Clarifications listed in this section apply to the following documents:

- *Pentium® II Xeon™ Processor at 400 and 450 MHz* datasheet
- *Intel Architecture Software Developer's Manual*, *Volumes 1, 2, and 3*
- *P6 Family of Processors Hardware Developer's Manual*

All Specification Clarifications will be incorporated into a future version of the appropriate Pentium II Xeon processor documentation.

D1. Thermal Sensor SMBus Address Latching

In the *Pentium® II Xeon™ Processor at 400 and 450 MHz* datasheet, Section 4.3.7, the following paragraph should be added:

> The thermal sensor latches the SA1 and SA2 signals at power up. System designers should ensure that these signals are at valid input levels (see Table 9) before the thermal sensor powers up. This should be done by pulling the pins to $V_{CCSMBus}$ or V_{SS} via a 1 kΩ or smaller resistor. Additionally, SA2 may be left unconnected to achieve the tri-state or "Z" state. If the designer desires to drive the SA1 or SA2 pin with logic the designer must ensure that the pins are at valid input levels (see Table 9) before $V_{CCSMBus}$ begins to ramp. The system designer must also ensure that their particular system implementation does not add excessive capacitance (>50 pF) to the address inputs. Excess capacitance at the address inputs may cause address recognition problems.

D2. MTRR Initialization Clarification

The following sentence should be added to the end of the first paragraph of Section 9.11.5 of the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*: "The MTRRs must be disabled prior to initialization or modification."

D3. Floating-Point Opcode Clarification

Section 3.2 of the *Intel Architecture Software Developer's Manual, Volume 2: Instruction Set Reference,* provides detailed descriptions of each Intel Architecture instruction. For some instructions, the clarification phrase below needs to be either added to their existing "Comments" section or a "Comments" section needs to be created with the clarification phrase. The phrase is as follows:

> The (**Instruction** shown in the center column of the table below) instruction is actually a combination of two instructions – the wait instruction followed by (**Instruction** shown in the table). If the (**Instruction** shown in the table) instruction should fault in some way (e.g., page fault), the value of EIP that is passed to the fault handler will be equal to the EIP of the first instruction plus one (i.e., the EIP of the second of the pair of instructions). The FWAIT portion of the combined instruction will have completed execution and will typically not be, nor need to be, re-executed after the fault handler is completed.

The following table lists the affected instructions and the location of the clarification phrase:

Instruction Set Reference Section	Opcode	Instruction	Clarification	Page
FCLEX/FNCLEX-Clear Exception	9B DB E2	FCLEX	Add "Comments" section with clarification phrase	3-177
FINIT/FNINIT-Initialize Floating-Point Unit	9B DB E3	FINIT	Add clarification phrase to existing "Comments" section	3-204
FSAVE/FNSAVE-Store FPU State	9B DD /6	FSAVE m94/108byte	Add clarification phrase to existing "Comments" section	3-237
FSTCW/FNSTCW-Store Control Word	9B D9 /7	FSTCW m2byte	Add "Comments" section with clarification phrase	3-250
FSTENV/FNSTENV-Store FPU Environment	9B D9 /6	FSTENV m14/28byte	Add "Comments" section with clarification phrase	3-253
FSTSW/FNSTSW-Store Status Word	9B DD /7	FSTSW m2byte	Add "Comments" section with clarification phrase	3-256

D4. PI-ROM S-Spec Clarification

The datasheet for the *Pentium® II Xeon™ Processor at 400 and 450 MHz* (CPUID 65*x*h) specifies a field in the PI-ROM for the "S-Spec/QDF Number". Production Pentium® II Xeon™ Processors at 400 and 450 MHz do not have the first byte of this field programmed, leaving it in the default state as a space (20H).

SPECIFICATION CHANGES

The Specification Changes listed in this section apply to the following documents:

- *Pentium® II Xeon™ Processor at 400 MHz and 450 MHz* datasheet
- *P6 Family of Processors Hardware Developer's Manual*
- *Intel Architecture Software Developer's Manual, Volumes 1, 2, and 3*

All Specification Changes will be incorporated into a future version of the appropriate Pentium II Xeon processor documentation.

D1. Locks Across Cache Line Boundary Disable Bit Added

In the Pentium II Xeon processor, setting bit 31 of the Model Specific Register (MSR) at address 33h to '1' will prevent LOCK# from being asserted when locked transactions which are split across a cache line boundary are issued from the processor. This bit is disabled by default and remains Reserved for all previous steppings of the Pentium II Xeon processor. In the default state ('0'), unaligned data issued in a locked sequence by the processor will have atomicity with the LOCK# signal asserted. When the bit is set, any transactions issued which split a cache line boundary will not have the LOCK# signal asserted, and no atomicity can be guaranteed between the reads and writes in the sequence. Locked sequences which do not split a cache line boundary will still follow the normal LOCK# protocol with this bit set.

D2. Non-GTL+ Output Leakage Current Change

The CMOS, TAP, Clock and APIC Signal Group Output Leakage Current specification (I_{LO} in Table 8 of the *Pentium® II Xeon™ Processor at 400 and 450 MHz* datasheet) should be changed from ±15 µA to ±30 µA.

D3. RESET# Pin Definition

The *P6 Family of Processors Hardware Developer's Manual* and the *Pentium® II Xeon™ Processor at 400 MHz and 450 MHz* datasheet both have incorrect definitions of the RESET# pin in their alphabetical signal listing. These documents incorrectly state:

> RESET# must remain active for one microsecond for a 'warm' Reset; for a Power-on Reset, RESET# must stay active for at least one millisecond after V_{CCCORE} and CLK have reached their proper specifications.

They should state:

> For a Power-on or 'warm' reset, RESET# must stay active for at least one millisecond after V_{CCCORE} and CLK have reached their proper specifications.